ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

Desarrollo de Técnicas de Aprendizaje Automático para la Predicción de Resultados de Partidos en Ligas Futbolísticas

Development of Machine Learning techniques for predicting results from League football matches

Realizado por
D. Francisco Javier Moreno Barea
Tutorizado por
D. Gracián Triviño Salas
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Septiembre de 2015

Fecha defensa:
El Secretario del Tribunal

Resumen

En la actualidad, existen un gran número de investigaciones que usan técnicas de aprendizaje automático basadas en árboles de decisión. Como evolución de dichos trabajos, se han desarrollado métodos que usan Multiclasificadores (Random forest, Boosting, Bagging) que resuelven los mismos problemas abordados con árboles de decisión simples, aumentando el porcentaje de acierto. El ámbito de los problemas resueltos tradicionalmente por dichas técnicas es muy variado aunque destaca la bio-informática. En cualquier caso, la clasificación siempre puede ser consultada a un experto considerándose su respuesta como correcta. Existen problemas donde un experto en la materia no siempre acierta. Un ejemplo, pueden ser las quinielas (1X2). Donde podemos observar que un conocimiento del dominio del problema aumenta el porcentaje de aciertos, sin embargo, predecir un resultado erróneo es muy posible. El motivo es que el número de factores que influyen en un resultado es tan grande que, en muchas ocasiones, convierten la predicción en un acto de azar.

En este trabajo pretendemos encontrar un multiclasificador basado en los clasificadores simples más estudiados como pueden ser el Perceptrón Multicapa o Árboles de Decisión con el porcentaje de aciertos más alto posible. Con tal fin, se van a estudiar e implementar una serie de configuraciones de clasificadores propios junto a multiclasificadores desarrollados por terceros. Otra línea de estudio son los propios datos, es decir, el conjunto de entrenamiento. Mediante un estudio del dominio del problema añadiremos nuevos atributos que enriquecen la información que disponemos de cada resultado intentando imitar el conocimiento en el que se basa un experto.

Los desarrollos descritos se han realizado en R. Además, se ha realizado una aplicación que permite entrenar un multiclasificador (bien de los propios o bien de los desarrollados por terceros) y como resultado obtenemos la matriz de confusión junto al porcentaje de aciertos.

En cuanto a resultados, obtenemos porcentajes de aciertos entre el 50% y el 55%. Por encima del azar y próximos a los resultados de los expertos.

Palabras claves

Aprendizaje automático, sistemas multiclasificadores, boosting, bagging, árbol de decisión, red neuronal, minería de datos, fútbol profesional, Liga BBVA, quiniela.

Abstract

Currently, there are a great number of investigations using machine learning techniques based on decision trees. As evolution of these works have developed methods using Multiclassifiers (Random forest, Boosting, Bagging) that solve the same problems addressed with simple decision trees, increasing the percentage of correct answers. The scope of the problems traditionally solved by these techniques are varied but stresses bio-informatics. In any case, the classification can always be consulted to an expert considered his answer as correct. There are problems where an expert in the field is not always succeeds. An example may be the pools (1X2). Where we can see that a knowledge of the problem domain increases the percentage of correct, however, predict an erroneous result is very possible. The reason is that the number of factors that influence an outcome is so great that, in many cases, makes the prediction in an act of chance.

In this paper, we try to find a multiclassifier based on the most studied simple classifiers such as the Multilayer Perceptron or decision trees with the highest possible percentage of successes. To this end, they will study and implement a number of configurations of own classifiers and multiclassifiers developed by third parties. Another line of research is the data itself, that is, the training set. Through a study of the problem domain we add new attributes that enrich the information that we have, trying to imitate each result in the knowledge that an expert is based.

The developments described have been made in R. Moreover, it has made an application to train a multiclassifier (either own or developed by others) and as result we obtain the confusion matrix by the percentage of correct answers.

As results, we get hit percentage between 50% and 55%. Above chance and next to the results of the experts.

Keywords

Computational learning, multi classifier systems, boosting, bagging, decision tree, neural network, data mining, professional football, Liga BBVA, football pools.

Índice

Resumen 5
Lista de Figuras 9
Lista de Tablas 9
1. Introducción 11
    1.1. Objetivos 12
    1.2. Estructura de la memoria 13
    1.3. Estado del Arte 14
2. Descripción del Problema 15
    2.1. Elección del Conjunto de Datos 16
    2.2. Descripción del Conjunto de Datos 19
3. Clasificadores 27
    3.1. Arboles de Decisión 27
    3.2. Redes Neuronales 29
    3.3. Multiclasificadores 30
        3.3.1. Arquitectura 31
        3.3.2. Métodos de combinación 33
    3.4. Random Forest 34
    3.5. Bagging 36
    3.6. AdaBoost.M1 37
4. Implementación de los multiclasificadores desarrollados 39
    4.1. Diseños externos 40
    4.2. Diseños propios 45
    4.3. Otros diseños 58
5. Análisis de resultados 59
    5.1. Precisión Predictiva 60
    5.2. Matriz de confusión 62
6. Conclusiones 65
Referencias 67
Anexo 69
    1. Instalación de R 69
    2. Paquetes R 70
    3. Aplicación: Presentación de los Datos 71
    4. Aplicación: Entrenamiento y Predicción 72

Lista de Figuras

Figura 1. Ingresos por derechos de televisión 16
Figura 2. Los 5 fichajes más caros de la historia 17
Figura 3. Número de títulos europeos desde el 2000 18
Figura 4. Clasificación de las instancias del problema 26
Figura 5. Ejemplo de árbol de decisión 28
Figura 6. Red neuronal artificial 29
Figura 7. Arquitectura en serie o vertical 31
Figura 8. Arquitectura en paralelo u horizontal 32
Figura 9. Arquitectura híbrida 32
Figura 10. Importancia Gini obtenida de Random Forest 35
Figura 11. Precisión predictiva con los diferentes conjuntos 61
Figura 12. Diferencia en el porcentaje de acierto 62
Figura 13. Matriz de confusión de diferentes modelos 63
Figura 14. Matriz de confusión de TFG 33 de forma gráfica 64
Figura 15. Interfaz de desarrollo de RStudio 69
Figura 16. Ejemplo de ejecución de la aplicación AppData 71
Figura 17. Ejemplo de ejecución de la aplicación AppQuiniela - Pruebas 72
Figura 18. Ejemplo de ejecución de la aplicación AppQuiniela - Quinielas 73

Lista de Tablas

Tabla 1. Discretización de los presupuestos anuales 20
Tabla 2. Pesos para cada clasificador individual 55
Tabla 3. Medidas de evaluación obtenidas 60

1. Introducción

En este trabajo pretendemos resolver el problema de la predicción de resultados de partidos de una liga de fútbol profesional. Los resultados que deseamos predecir no son numéricos sino un resultado más simplificado y que es utilizado por un juego de "*Loterías y Apuestas del Estado*": los resultados de los partidos en formato quiniela.

Desde el punto de vista de la predicción de los resultados de los partidos de fútbol, el problema que nos ocupa es complejo. En ellos suelen influir variables singulares, como la ausencia de un jugador estrella o la situación de un nuevo entrenador en el equipo, que no se encuentran presentes en históricos, y no son sencillas de recoger en el conjunto de datos usado para el entrenamiento del algoritmo de aprendizaje.

La predicción es difícil incluso para un experto. Sin embargo, lo curioso es que una persona con conocimientos de fútbol pero no considerada experta es capaz de tener un acierto cercano al 40%, y si predijéramos siempre el resultado como 1, obtendríamos un acierto superior al 45%. Este hecho nos lleva a pensar que un algoritmo o técnica de aprendizaje adecuado pueda alcanzar tasas de acierto superiores al 50% como mínimo. La búsqueda de dicho algoritmo y su implementación es la tarea fundamental de este trabajo.

En resumen la formulación del problema es: dadas las instancias de los partidos que se van a jugar en una determinada jornada de la primera división española de fútbol, ser capaces de clasificarlos en las clases 1, X o 2 con el mayor porcentaje de acierto.

Para la resolución de este problema, se llevará a cabo un proceso KDD (del inglés *Knowledge Discovery in Databases*). Erróneamente se conoce a este proceso como minería de datos; sin embargo la minería de datos es en sí una fase del proceso [1]. Para ello se seguirán sus fases [2]: una fase de recopilación y preparación de los datos que van a ser utilizados; una fase en la cual se estudiarán y diseñarán técnicas y modelos de aprendizaje automático; se procede con la fase de obtención de patrones y resultados a partir de las técnicas anteriores y los datos recopilados; y por último se evalúa e interpreta el conocimiento obtenido, con su posterior divulgación.

Como se ha dicho, una de las fases que se van a llevar a cabo, es el estudio de diferentes técnicas de aprendizaje automático. El trabajo se centra en las técnicas que utilizan multiclasificadores. Una breve definición de multiclasificador es, conjunto de clasificadores que obtiene como resultado la fusión o la combinación de las predicciones de los mismos [3]. Se pueden diseñar diferentes multiclasificadores atendiendo a las características que presentan, como el número de clasificadores individuales utilizados, el tipo de clasificador o el tamaño de los datos que utiliza cada clasificador [4] [5]. De esta forma, se tienen varias arquitecturas posibles y diferentes estrategias de combinación de las predicciones de cada clasificador [3].

Entre los multiclasificadores descritos en la literatura, serán usados algunos de ellos, como son Bagging [6] [7], AdaBoost.M1 [7] [8] o Random Forest [9]. A continuación, se diseñaran un cierto número de multiclasificadores combinando clasificadores clásicos, como pueden ser los árboles de decisión o las redes neuronales artificiales, con el propósito de comparar la tasa de resultados acertados obtenida con las técnicas anteriormente estudiadas y con los modelos diseñados, y comprobar si estos últimos, creados a medida para los datos, obtienen una tasa mayor. Las implementaciones de dichas técnicas se realizarán en el lenguaje estadístico R.

Las novedades principales, como parte de motivación del trabajo, se resumen en:

1. El campo de las técnicas que usen sistemas multiclasificadores es un campo del aprendizaje automático relativamente nuevo. Hay poca literatura sobre él, y la que hay no es anterior los años 90. Sin embargo, es un campo que tiene mucho potencial, ya que según diversos estudios realizados, el resultado obtenido por la combinación de clasificadores es mejor que el obtenido por el mejor clasificador [4]. Gracias al uso de un método de combinación (como la votación), o de un meta-nivel clasificador con los resultados obtenidos de otros clasificadores como entrada, se consiguen en teoría, resultados mejores que el mejor resultado obtenido de los clasificadores base.

2. La mayor parte de algoritmos, técnicas y programas utilizados en este campo del aprendizaje automático están diseñados para resolver problemas del ámbito de la medicina o la bio-investigación. Por ello, la aplicación de estos en un problema tan peculiar como el que atañe a este trabajo es una novedad, ya que se intenta predecir un acontecimiento deportivo. Habitualmente, los distintos sistemas creados con el propósito de resolver este problema se basan únicamente en métodos estadísticos como puede ser el uso de distribuciones discretas [10] [11]. Además, recordar los beneficios empresariales que un algoritmo de esta índole pueda llegar a generar [13].

1.1. Objetivos

Este trabajo se centrará en los siguientes aspectos:

- Descripción del problema de la predicción de resultados de partidos en ligas futbolísticas, de su problemática, y del conjunto de datos que va a ser usado para su resolución.
- Razonar la elección de la liga española de fútbol como fuente de nuestros datos para la realización del trabajo.
- Estudio de diferentes técnicas de aprendizaje automático que van a ser utilizadas para la resolución del problema. Se centrará en las técnicas que utilizan multiclasificadores y su diseño.
- Identificación de una posible mejora de los métodos existentes para la resolución del problema, implementando las técnicas y modelos diseñados.

- Análisis de los resultados obtenidos tras la aplicación de nuestros conjuntos de datos con las técnicas y modelos diseñados, y comparación de resultados entre los obtenidos con estos y con los métodos estudiados.

- Realizar un aporte de carácter investigador con las conclusiones finales e identificar posibles líneas de investigación futuras.

1.2. Estructura de la memoria

La presente memoria se divide en los siguientes apartados:

- En el capítulo 2, ***Descripción del problema***, se describe el problema que se trata en el trabajo. En primer lugar, se explica cómo se enfoca el problema de la predicción de resultados de partidos de fútbol. Posteriormente, se describen las razones por las cuales ha sido elegida la liga de fútbol profesional española como fuente de los datos para el problema, en detrimento de otras ligas europeas de fútbol profesional. Por último se detallan los atributos que presentan los datos, explicando el dominio de los mismos y empezando a interpretar la importancia que tienen a la hora de predecir un partido.

- El capítulo 3, ***Clasificadores***, se centra en el estudio de los diferentes algoritmos o métodos del aprendizaje automático que van a ser utilizados en los siguientes capítulos. Serán definidos y se explicará cómo se crea cada modelo a partir de los datos. Una parte esencial de este capítulo es la descripción de las diferentes arquitecturas y métodos de combinación que presentan los multiclasificadores, que serán utilizados en el siguiente capítulo.

- En el capítulo 4, ***Implementación de los multiclasificadores desarrollados***, serán presentados los modelos multiclasificadores que han sido diseñados para la resolución del problema buscando una mejor predicción que la de las técnicas previamente analizadas. Además se verá su división en ciertos grupos según su estructura, y se describirán las implementaciones de estos y de los modelos presentados en el capítulo anterior que también son utilizados.

- En el capítulo 5, ***Análisis de resultados***, se describe el sistema de pruebas y evaluación utilizada, así como las diferentes medidas de evaluación del sistema. Estas pruebas se realizan con el fin de poder comprobar la precisión de los modelos desarrollados a la hora de resolver el problema. Además, se hará una comparación entre los resultados obtenidos con los métodos descritos en el capítulo 3 y los modelos diseñados en el trabajo, para comprobar si se ha conseguido una mejora respecto a los primeros.

- En el capítulo 6, ***Conclusiones***, se concluye la memoria del trabajo realizado, extrayendo unas conclusiones sobre el trabajo llevado a cabo.

1.3. Estado del Arte

En los últimos años ha habido un crecimiento imparable de las casas de apuestas deportivas. Según un artículo del periódico "*20 Minutos*" [13], en el año 2008 la inversión realizada en el ámbito de las apuestas deportivas en la región de Madrid, rozaba los 20.6 millones de euros, un 0.5% del total invertido en juegos de azar. En 2012, esta inversión se disparó hasta alcanzar un volumen de 191 millones de euros, un incremento del 827%. Gracias a un vacío legal debido a su reciente aparición en la vida pública, las casa de apuestas han logrado posicionarse por encima los locales tradicionales presenciales (salas físicas de casino, bingo). Además de esto, han inundado las camisetas de los equipos de fútbol con su publicidad y han lanzado anuncios en radio y televisión, por lo que su poder entre la población ha aumentado.

La mayor parte de estas casas de apuestas deportivas son sitios web. Entre todas las apuestas deportivas, las más demandadas son las referentes al fútbol profesional. Para medir el valor de las apuestas, la mayoría utilizan métodos estadísticos para la predicción de acontecimientos [10], en detrimento de los métodos de aprendizaje automático utilizados en la minería de datos, estos últimos utilizados en el trabajo. Dos de los métodos estadísticos más utilizados para este tipo de apuestas son la distribución discreta bivariada [11] y el sistema de puntuación ELO [12].

- En el caso de la **distribución discreta bivariada**, los modelos empleados se definen en términos de las distribuciones marginales y una cópula, y se utilizan para describir los resultados de los partidos. Cópula [11] es una distribución con todas las distribuciones marginales distribuidas uniformemente en el intervalo [0,1]; es decir, es la distribución de un vector aleatorio uniforme.

  Esta representación de la cópula, permite a la distribución bivariante ser modelada de una manera flexible mediante la especificación de una familia adecuada de las funciones cópula y ajustando esto a los datos de dos variables utilizando máxima verosimilitud.

- El **sistema de puntuación ELO** es un método para el cálculo de los niveles de habilidad relativos de los jugadores en juegos jugador vs. jugador. Fue diseñado como un sistema para el ajedrez, pero actualmente se utiliza como sistema de clasificación en videojuegos, baloncesto, béisbol o fútbol [12].

  La diferencia de puntuación entre dos jugadores sirve como predictor del resultado de un partido. La puntuación de un jugador aumenta o disminuye en base a los resultados de los partidos, el jugador ganador obtiene puntos del perdedor. Si el jugador con alta puntuación gana, se le dan pocos puntos del perdedor. Sin embargo, si el jugador con baja puntuación consigue una inesperada victoria, se le transfieren muchos puntos. Esto significa que el sistema de calificación es de auto-corrección.

2. Descripción del Problema

Como ya ha sido mencionado, nuestro problema es conseguir predecir el resultado de los diferentes partidos de una liga de fútbol profesional.

Para hacer más general el problema, y poder enfocarlo de alguna forma al mundo de las quinielas, que mueve millones de euros, en lugar de predecir el resultado numérico de un partido de fútbol profesional, queremos predecir el resultado del partido en formato quiniela. En estos resultados tendremos que el 1 significa que gana el equipo que juega como local, 2 que gana el equipo que juega como visitante, y X que los equipos empatan.

El problema elegido es complejo. Los resultados de partidos de fútbol son difíciles de predecir y la información que tenemos de ellos es limitada. Antes de seguir, vamos a explicar que queremos decir con que la información es limitada. La mayor carga de trabajo cuando se realiza un proceso KDD [1] se centra en la recolección y el tratamiento de los mismos. En este sentido, la mayor parte de los datos que podemos recoger de una liga de fútbol profesional son de dos tipos, o bien son tablas de clasificación de final de temporada, con atributos como los puntos conseguidos, así como los goles marcados y recibidos y la diferencia de los mismos; o bien, son partidos jornada por jornada de los que solo se puede recabar la temporada y la jornada en los que se jugó el partido, los equipos que lo jugaron, y su resultado final.

Los datos recolectados para este trabajo son del segundo tipo, ya que el problema es la predicción de estos resultados. Antes de trabajar con ellos, ha de hacerse un tratamiento de los datos, y así agregar nuevos atributos a los anteriormente dichos. Estos atributos son los que un experto necesitaría para una predicción más acertada del resultado, como son los presupuestos de cada equipo, como quedaron la temporada anterior en la clasificación, en el momento que se jugó el partido los puntos que llevaba cada uno, goles tanto a favor como en contra, racha de partidos, gol average de ambos equipos, etc.

Aun así, los resultados de los partidos de fútbol pueden depender de situaciones que son difíciles de recoger de partidos pasados, como pueden ser la ausencia de un jugador estrella, la situación de un nuevo entrenador en el equipo u otros factores que llevan a que un equipo que en principio debiera ganar, pierda. Este tipo de información es de gran utilidad, pero de muy difícil acceso si es referente a temporadas anteriores a la actual. Por lo tanto, en el trabajo no serán utilizados atributos relacionados al estado de forma de jugadores, a la ausencia de los mismos, o a información referente al entrenador.

El problema se puede resumir pues en, dadas las instancias de diferentes partidos de la primera división española de fútbol, ser capaces de clasificarlos en las clases 1, X o 2 con el mayor porcentaje de acierto.

2.1. Elección del Conjunto de Datos

Para este trabajo, la liga de fútbol profesional elegida ha sido la liga de primera división española, llamada Liga BBVA desde la temporada 2008/2009, cuando se llegó a un acuerdo de patrocinio con la conocida entidad bancaria.

La elección de esta liga profesional viene dada por diversas razones:

- Es la **liga de fútbol profesional de nuestro país**. Hoy en día, la mayor parte de la gente corriente conoce, aunque sea de forma mínima, equipos o estrellas de esta, cosa que no ocurriría con ligas de fútbol de otros países como la Serie A o la Bundesliga (ligas de fútbol de Italia y Alemania respectivamente). En nuestro país, el fútbol está inmerso en la cultura y muchos aspectos de la vida giran en torno a él. Por esta razón algunas de las conclusiones que se obtengan en este trabajo pueden ser interesantes para el público en general.

- Es una de las ligas que **más beneficios genera** por patrocinios. Esto es a excepción de la Premier League inglesa, la cual posee un reparto televisivo equitativo, lo que provoca que equipos modestos puedan disponer de un mayor presupuesto, moviendo más el mercado de fichajes.

Figura 1. Ingresos por derechos de televisión

En la liga española, Real Madrid y FC Barcelona rechazan el modelo inglés. El modelo actual en España es el reparto entre Madrid y Barcelona del 50 por ciento del dinero generado por las cadenas de televisión. Esto significa una diferencia de 11 a 1 con respecto al resto de equipos de la liga.

- Es la liga de fútbol profesional con los dos **equipos de fútbol más valiosos del mundo** según la revista Forbes [14]. De esta forma, el FC Barcelona con $3.16 billones es segundo de esta lista, encabezada por el Real Madrid con $3.26 billones. El tercero de la lista es el Manchester United, de la Premier League inglesa, con $3.1 billones gracias al reciente contrato que ha realizado con la marca de ropa deportiva Adidas. Aquí es donde se rompe la competición entre los equipos de la lista, ya que el siguiente equipo es el FC Bayern, de la Bundesliga alemana, con $2.35 billones, y el siguiente es el Manchester City, de la Premier League, con $1.38 billones.

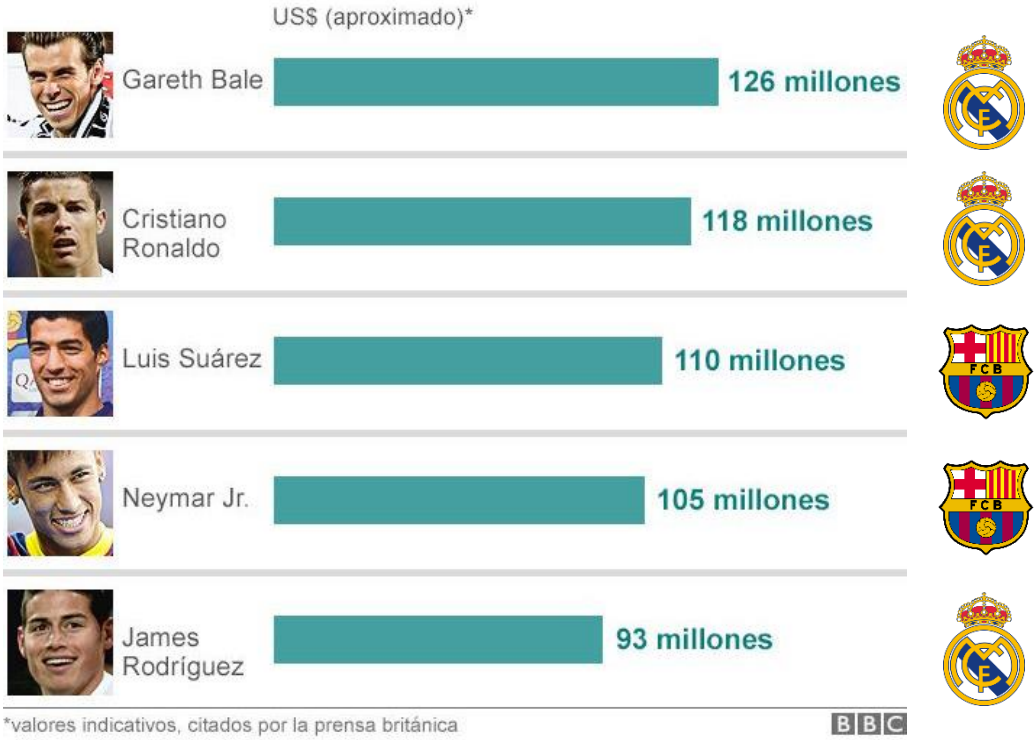

Figura 2. Los 5 fichajes más caros de la historia

- Es la liga **más exitosa en los últimos 15 años** en competiciones europeas. La diferencia entre ligas es tan grande que la suma de los títulos conseguidos por los otros tres países no alcanza el total logrado por los equipos españoles.

  El FC Barcelona, con cuatro Ligas de Campeones, y el Sevilla con cuatro Copas UEFA/Liga Europa, encabezan una lista que también integran Real Madrid (tres Ligas de Campeones), Atlético de Madrid (dos UEFA) y Valencia (una UEFA). Inglaterra ocupa el segundo lugar al levantar tres Ligas de Campeones (ganadas por Liverpool, Manchester United y Chelsea) y dos títulos en la copa UEFA/Liga Europa (Liverpool y Chelsea).

  Lejos queda la década de los 90, en la que los equipos italianos (Juventus, AC Milán e Inter de Milán) dominaron el continente con tres títulos en la Copa de Europa/Champions y siete en la Copa UEFA.

Figura 3. Número de títulos europeos desde el 2000

- Es la liga **más competitiva** objetivamente de las grandes ligas europeas. Se suele decir que la liga española es un campeonato de dos equipos, Real Madrid y FC Barcelona, que como hemos mencionado anteriormente, son los equipos de fútbol más valiosos del mundo. Se puede decir que en términos de ganadores de la competición, esto es cierto, ya que en los últimos 10 años, 9 veces han ganado la competición uno de estos dos equipos y 9 veces han sido subcampeones. Sin embargo, la liga española es la más abierta e impredecible, al haber hasta 12 equipos diferentes que lograron clasificar por lo menos una vez entre los cuatro primeros de la competición (que dan acceso a la Liga de Campeones) en la última década. También si tenemos en cuenta que, por ejemplo, este año, a tres jornadas de terminar la competición no había campeón (en Inglaterra y Alemania sí), y solo había un descendido seguro, con hasta 6 equipos que todavía tenían posibilidades de descender.

- Por último, en España tenemos el juego de azar de *Loterías y Apuestas del Estado*, **"*La Quiniela*"** que trata nuestro problema. Se les ha pedido opinión a un número reducido de loteros sobre, según su experiencia, el porcentaje de acierto que obtienen los jugadores. La opinión generalizada es, que una persona normal que tenga un conocimiento estándar de la Liga BBVA, puede llegar a obtener un acierto del 40% a lo largo de toda la temporada, y que una persona a la que podamos llegar a considerar experta en el tema, puede llegar a obtener un acierto del 50% de media, por lo que ya tenemos estadísticas iniciales sobre el acierto que un humano obtiene en el problema.

2.2. Descripción del Conjunto de Datos

El conjunto de datos recabados para el trabajo son todas las jornadas que se han jugado en la Liga BBVA desde la temporada 2004/2005 a la temporada 2014/2015 (la temporada que acabó en Mayo de este año). Hemos decido recoger los datos de estas temporadas para el trabajo debido a un suceso en común, Real Madrid o FC Barcelona son campeón o subcampeón. En estas 11 temporadas, estos dos equipos se repartieron 10 campeonatos (7 FCB/3 RM) y 10 subcampeonatos (7 RM/3 FCB). En las 5 temporadas anteriores a estas, solo fue campeón 2 veces el Real Madrid y 2 veces subcampeón el FC Barcelona.

Los atributos de las instancias utilizadas en el trabajo son:

- **Temporada (*temp*):**
  Atributo nominal que representa la temporada en la cual tiene lugar el encuentro entre ambos equipos. Como ya se ha mencionado, los datos agrupan los encuentros desde la temporada 2004/2005 a la 2014/2015, por lo que los representamos de la siguiente forma: *04/05, 05/06,…, 13/14, 14/15*.

- **Jornada (*jrd*):**
  Atributo numérico que indica la jornada en la cual tiene lugar el encuentro. Cada temporada está configurada en 38 jornadas, ya que son 20 equipos y cada equipo ha de jugar con el resto dos veces, una como local y una como visitante. Por esta razón, este atributo va desde 1 a 38.

- **Equipo local (*eqLocal*):**
  Es el atributo nominal correspondiente a los nombres de los equipos de la liga que juegan el correspondiente partido como local. Jugar como local significa que el equipo juega en su estadio, en su ciudad, con la mayor parte del público a su favor y apoyándole, por lo que la moral del equipo en esta situación suele aumentar, lo que hace que sus probabilidades de ganar aumenten. Por esta razón hemos querido reflejar que equipo es local y cual es visitante, y no poner un simple encuentro entre ambos.

  Los nombres de los equipos han tenido que ser unificados debido a que hay diversas formas de poder referirse al mismo equipo. Así por ejemplo, para referirnos al Atlético de Madrid SAD, podemos hacerlo por A. Madrid, Atlético o At. Madrid. Hay 37 equipos distintos unificados de la siguiente forma:

  Alaves, Albacete, Almeria, At. Madrid, Athletic, Barcelona, Betis, Cadiz, Celta, Cordoba, Deportivo, Eibar, Elche, Espanyol, Getafe, Gimnastic, Granada, Hercules, Levante, Malaga, Mallorca, Murcia, Numancia, Osasuna, Racing, Rayo Vallecano, Real Madrid, Real Sociedad, Recreativo, Sevilla, Sporting, Tenerife, Valencia, Valladolid, Villareal, Xerez, Zaragoza.

- **Puntos totales del equipo local (*pTotalLocal*):**
  Atributo numérico que indica los puntos que llevaba el equipo local antes de que se produjera el encuentro. En nuestros datos el mínimo de puntos es 0, ya que en la primera jornada todos los equipos empiezan con 0 puntos en el casillero. El máximo de puntos alcanzado en los datos es de 97 antes de concluir la última jornada de liga. La media de puntos conseguidos por el equipo local es de 25.3 puntos.

  Los puntos que se dan a un equipo vienen dados por el resultado del encuentro. Si el equipo ha ganado el partido, se le otorgan 3 puntos; si el partido acaba en empate, se le otorga 1 punto; y si ha acabado en derrota, no se le otorgan puntos. Antes de Octubre de 1994, al equipo que ganaba el encuentro se le daban 2 puntos, pero en esta fecha, la FIFA tomo la decisión de cambiar el sistema de puntuación otorgando 3 puntos al vencedor del encuentro. De esta forma, alegaban que se producirían menos empates al haber más diferencia de puntos con la victoria.

- **Presupuesto del equipo local (*presLocal*):**
  Atributo nominal que representa el presupuesto del equipo local. Los equipos suelen disponer de un presupuesto para la temporada, como cualquier empresa, para hacer nuevos fichajes, pagar sueldos o para mantenimiento. Estos presupuestos suelen estar en formato numérico, aunque no son totalmente fiables. Para darle más fiabilidad decidimos discretizarlos, de esta forma tenemos que este atributo puede tomar los valores: *Muy Alto, Alto, Medio, Bajo, Muy Bajo*.

  La forma de discretizarlos varía un poco según la temporada, ya que según avanzamos en el tiempo, los equipos disponen cada vez de un presupuesto más elevado. De este modo lo que hacemos para cada temporada es buscar una media de los presupuestos, que varía entre 35 M€ y 40 M€, y partiendo de esta medida discretizamos los presupuestos. En la siguiente tabla disponemos del estándar que hemos utilizado:

Discretización	Presupuesto Numérico
Muy Alto	Más de 90 M€
Alto	Entre 45 M€ y 90 M€
Medio	Entre 30 M€ y 45 M€
Bajo	Entre 20 M€ y 30 M€
Muy Bajo	Menos de 20 M€

Tabla 1. Discretización de los presupuestos anuales

- **Goles a favor del equipo local (*gFavorLocal*):**
  Este atributo numérico representa el número de goles a favor del equipo local antes de disputarse el encuentro. En este atributo se recogen todos los goles que ha logrado marcar el equipo local, ya sea en partidos que ha jugado como local o en partidos que haya jugado como visitante. Al inicio de la temporada los goles a favor del equipo son 0, por tanto este es el mínimo. El máximo de goles a favor en nuestros datos es de 117 antes de disputarse la última jornada. La media de goles a favor del equipo local es de 24.5 goles.

- **Goles en contra del equipo local (*gContraLocal*):**
  Este atributo numérico indica el número de goles en contra que ha recibido el equipo local antes de disputarse el encuentro. En este atributo se recogen todos los goles que ha encajado el equipo local, ya sea en partidos que ha jugado como local o en partidos que haya jugado como visitante. Al inicio de la temporada los goles en contra del equipo son 0, por tanto este es el mínimo. El máximo de goles en contra en nuestros datos es de 84 antes de disputarse la última jornada. La media de goles a favor del equipo local es de 24.7 goles.

- **Diferencia de goles del equipo local (*difGlsLocal*):**
  Atributo numérico que indica la llamada diferencia de goles, o gol average, del equipo local antes de disputarse el encuentro. Este atributo se calcula restando a los goles a favor, los goles en contra, de esta forma tenemos que este atributo puede tomar valores positivos si el equipo ha metido más goles de los que ha encajado; o negativos si el equipo ha encajado más goles que los que ha conseguido meter. Este atributo es un buen indicativo del estado de forma del equipo, ya que reúne en uno el estado de forma de sus atacantes (goles a favor) y el estado de forma de sus defensores (goles en contra).

  El mínimo que presenta este atributo en nuestros datos es de -42 y el máximo es de 89. Este mínimo y máximo, al contrario que en los dos anteriores atributos, no tienen por qué darse en la última jornada de liga, ya que no es un atributo acumulativo.

- **Clasificación del equipo local en la temp. anterior (*tAnteriorLocal*):**
  Atributo nominal que discretiza la posición en la que el equipo local quedo en la clasificación de la temporada anterior a en la que se produce el encuentro. De esta forma podemos saber si el equipo fue campeón o subcampeón de la temporada anterior, si estaba luchando por puestos europeos (del 1º al 6º) o luchando por no descender (16º y 17º). También indicamos si el equipo acaba de ascender con el valor *RA*, de “recién ascendido”.

  Los valores que toma son: *[1-2], [3-4], [5-6], [7-8], [9-15], [16-17], RA.*

- **Racha del equipo local (*rachaLocal*):**
  Atributo numérico que representa el estado de estado de forma a corto plazo, indicando la racha que lleva el equipo local. Este atributo ha sido calculado, ya que no se puede encuentra como tal en ningún histórico.

  Para calcularlo se ha seguido el siguiente algoritmo en pseudocódigo:

  SI resultado es *"Ganador"*
  ENTONCES
  SI rachaEquipoLocal >= 0
  ENTONCES rachaEquipoLocal = rachaEquipoLocal + 1
  SI NO rachaEquipoLocal = 0

  SI resultado es *"Perdedor"*
  ENTONCES
  SI rachaEquipoLocal >= 0
  ENTONCES rachaEquipoLocal = 0
  SI NO rachaEquipoLocal = rachaEquipoLocal – 1

  SI resultado es *"Empate"*
  ENTONCES
  SI rachaEquipoLocal > 1
  ENTONCES rachaEquipoLocal = rachaEquipoLocal – 2

  SI rachaEquipoLocal < -1
  ENTONCES rachaEquipoLocal = rachaEquipoLocal + 1

  Podemos apreciar del pseudocódigo ciertas situaciones:
  - **Ganar**:
    - Si el equipo llevaba una racha ganadora, le premiamos con una victoria más aumentando su racha.
    - Si llevaba racha perdedora, ponemos que se ha recuperado de su mala racha, por lo que su racha actual será nula (0).
  - **Perder**:
    - Si el equipo llevaba una racha ganadora, ponemos que se ha cortado su racha, por lo que pasa a ser nula (0).
    - Si llevaba racha perdedora, le perjudicamos con una derrota más, disminuyendo su racha con derrotas.
  - **Empatar**:
    - Si el equipo llevaba una racha ganadora, le perjudicamos por haber empatado, pero no le cortamos la racha, ya que suele ser un resultado puntual.
    - Si llevaba racha perdedora, le premiamos por haber empatado pero no cortamos su racha negativa, ya que el equipo local se suele encontrar con la mayor parte del público a su favor, lo que debe aumentar la moral del equipo.

- **Puntos en casa del equipo local (*pLocalCasa*):**
  Atributo numérico, parecido al atributo de "*Puntos totales del equipo local*", difiere de este en que indica los puntos que ha conseguido el equipo local sólo en partidos en los que ha jugado como local antes de que se produjera el encuentro. Este atributo es una buena medida de como de bueno es el equipo jugando en casa. Podemos poner el ejemplo de dos equipos, Numancia y Osasuna, que, aun siendo equipos de media tabla que al final descendieron, conseguían la mayor parte de sus puntos en los partidos que jugaban como local, dando lugar a lo que en el mundo del fútbol se conoce como "*fortines*", estadios en los que es difícil ganar como visitantes.

  En nuestros datos el mínimo de puntos es 0, ya que en la primera jornada todos los equipos empiezan con 0 puntos en el casillero. El máximo de puntos alcanzado en los datos es de 52 antes de concluir la última jornada de liga. La media es de 15 puntos.

- **Goles a favor en casa del equipo local (*gFavorLocalCasa*):**
  Al igual que ocurre con el atributo anterior, es una concretización de otro de los atributos vistos anteriormente. Este es un atributo numérico que representa al subconjunto de los goles a favor totales del equipo local. Así tenemos que este atributo indica el número de goles a favor que ha logrado meter un equipo jugando como local antes del encuentro.

  Puede pasar que el número de goles a favor totales sea muy elevado, pero el número de goles como local sea bajo, aunque la mayoría de las veces ocurre al contrario. Al inicio de la temporada los goles a favor del equipo jugando como local son 0, por tanto este es el mínimo. El máximo de goles a favor nuestros datos es de 69 antes de disputarse la última jornada. La media de goles a favor del equipo local como local es de 14 goles.

- **Goles en contra en casa del equipo local (*gContraLocalCasa*):**
  Este atributo numérico es el último atributo que se refiere exclusivamente al equipo local. Al igual que ocurre con los dos atributos anteriores, es muy parecido a otro atributo visto anteriormente. En este caso tenemos el subconjunto de los goles en contra totales del equipo local. Así tenemos que este atributo indica el número de goles en contra encajado por un equipo jugando como local antes del encuentro.

  Este atributo está también relacionado con el número de puntos obtenidos jugando como local, ya que como ocurre con este, suele ser un indicador de lo bueno que es el equipo jugando en casa, especialmente con grandes equipos que suelen ser siempre los que ganan con mayores goleadas incluso jugando como visitante. 0 es el mínimo, siendo 35 el máximo en nuestros datos antes de disputarse la última jornada. La media de goles en contra del equipo local como local es de 10 goles.

- **Equipo visitante (*eqVisit*):**
  Es el atributo nominal correspondiente a los nombres de los equipos de la liga que juegan el correspondiente partido como visitante. Los valores que puede tomar este atributo son los mismos que los que toma el equipo local.

- **Puntos totales del equipo visitante (*pTotalVisit*):**
  Atributo numérico que indica los puntos que llevaba el equipo visitante antes de que se produzca el encuentro. El mínimo de puntos es 0, ya que en la primera jornada todos los equipos empiezan con 0 puntos. El máximo de puntos alcanzado es de 96 antes de concluir la última jornada. La media de puntos conseguidos por el equipo visitante es de 25.6 puntos, la cual, como es lógico ya que son los mismos equipos, no varía demasiado de la media de puntos totales de los equipos locales.

- **Presupuesto del equipo visitante (*presVisit*):**
  Atributo nominal que representa el presupuesto del equipo visitante. Los valores que puede tomar son los mismos que en los presupuestos de los equipos locales y la discretización es la misma.

- **Goles a favor del equipo visitante (*gFavorVisit*):**
  Atributo numérico que indica el número de goles a favor del equipo visitante antes de disputarse el encuentro. El mínimo es 0, al comenzar la temporada, y el máximo alcanzado en nuestros datos es de 115 antes de disputarse la última jornada. La media de goles a favor del equipo local es de 24.7 goles.

- **Goles en contra del equipo visitante (*gContraVisit*):**
  Este atributo numérico indica el número de goles en contra que ha recibido el equipo visitante antes de disputarse el encuentro. Al inicio de la temporada es 0, por tanto este es el mínimo, y el máximo de goles es de 83. La media de goles a favor del equipo local es de 24.4 goles.

- **Diferencia de goles del equipo visitante (*difGlsVisit*):**
  Atributo numérico que indica la llamada diferencia de goles del equipo visitante antes de disputarse el encuentro. El mínimo que presenta este atributo en nuestros datos es de -43 y el máximo es de 88. Este mínimo y máximo, al contrario que en los dos anteriores atributos, no tienen por qué darse en la última jornada de liga, ya que no es un atributo acumulativo; pero al igual que ellos, los mínimos, máximos y media son similares a los del equipo local. Esto es lógico ya que son medidas generales. Todos los equipos han de jugar 19 partidos de local y 19 de visitante, intercalándose aleatoriamente los partidos, por lo que las medidas no cambian lo suficiente de unos a otros.

- **Racha del equipo visitante (*rachaVisit*):**
  Atributo numérico que representa el estado de estado de forma a corto plazo, indicando la racha que lleva el equipo visitante. Se produce un cambio respecto al cálculo de la racha para el equipo local, a la hora de un empate.

  - Si el equipo llevaba una racha ganadora, al equipo local le perjudicábamos por haber empatado restándole 2 a su racha. En este caso, al equipo visitante le restamos solo 1, ya que es más difícil como visitante empatar que perder.

  - Si llevaba racha perdedora, al equipo local le premiábamos por haber empatado pero no cortábamos su mala racha, sumándole 1 a su racha. La racha del equipo visitante, sin embargo, se incrementa en 2.

- **Clasificación del equipo visitante en la temp. anterior (*tAnteriorVisit*):**
  Atributo nominal que discretiza la posición en la que el equipo visitante quedo en la clasificación de la temporada anterior a la del encuentro.
  Los valores que toma este atributo son los mismos que para el equipo local, ya que son únicos para un equipo en una temporada, independientemente de que juegue como local o como visitante.

- **Puntos fuera del equipo visitante (*pVisitFuera*):**
  Atributo numérico, parecido al atributo de "*Puntos totales del equipo visitante*", difiere de este en que indica los puntos que ha conseguido el equipo visitante sólo en partidos en los que ha jugado como visitante antes de que se produjera el encuentro. De este modo podemos averiguar como de bueno es el equipo en un campo hostil, en la que la mayor parte del público este en su contra.

  En nuestros datos el mínimo de puntos es 0, por la primera jornada. El máximo de puntos alcanzado en los datos es de 47 antes de concluir la última jornada de liga. La media es de 9 puntos. Si las comparamos con las medidas obtenidas para los puntos en casa del equipo local, podemos observar como el máximo es 5 puntos inferior y la media menor en 6 puntos.

- **Goles a favor fuera del equipo visitante (*gFavorVisitFuera*):**
  Atributo numérico que representa al subconjunto de los goles a favor totales del equipo visitante. Sólo cuentan los que hayan logrado en partidos jugados como visitante.

  Al inicio de la temporada los goles a favor del equipo jugando como local son 0, por tanto este es el mínimo. El máximo de goles a favor nuestros datos es de 50 antes de disputarse la última jornada. La media de goles a favor es de 10 goles. En comparación con los goles que se logran como local, tenemos que son 19 goles menos de máxima y una media de 4 goles inferior.

- **Goles en contra fuera del equipo visitante (*gContraVisitFuera*):**
  Atributo numérico que representa al subconjunto de los goles en contra totales del equipo visitante. Sólo cuentan los encajados en partidos jugados como visitante. El mínimo es 0, siendo 48 el máximo en nuestros datos. La media de goles en contra del equipo visitante como visitante es de 14 goles. En comparación con las medidas de goles en contra del equipo local como local, tenemos un máximo 13 goles superior y una media de 4 goles más.

  Gracias a los tres atributos anteriores y a las comparativas con sus homólogas de los equipos jugando como local, se puede afirmar que jugar como visitante altera de cierto modo la forma de jugar de los equipos haciendo que sea más asequible ganar para el equipo que juega como local, que para el que juega como visitante.

- **Resultado del partido (*quiniela*):**
  Este es la clase que queremos predecir. El nombre que le hemos dado en el conjunto de datos es quiniela ya que, como ya se ha explicado, queremos predecir el resultado del encuentro en formato quiniela, y no el resultado numérico del partido. De este modo tendremos la clase "*1*", la clase "*2*" y la clase "*X*". La clase *1* significa que gana el equipo que juega como local, *2* que gana el equipo que juega como visitante, y *X* que los equipos empatan.

  Como podemos ver en la siguiente gráfica, el número de instancias clasificadas como *1* es casi el 50% de las instancias totales, lo que corrobora la afirmación de que ganar siendo local es más probable. También podemos observar que el número de instancias de la clase *2* es superior a las de la clase *X*, aun así, no están tan dispares como se podría pensar.

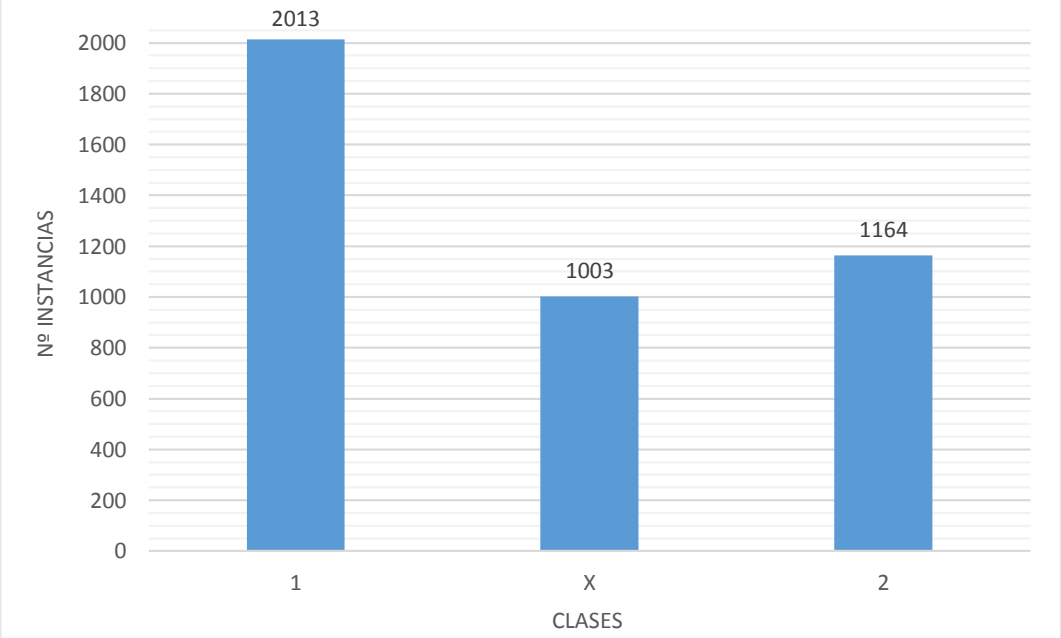

Figura 4. Clasificación de las instancias del problema

3. Clasificadores

En el proceso KDD, nos encontramos con la tarea de minería de datos, nombre con el cual nos referimos comúnmente a todo el proceso KDD [1]. Realmente, la tarea de minería de datos trata solamente el análisis automático, o semi-automático, de grandes conjuntos de datos con el objetivo de obtener a partir de ellos patrones desconocidos, grupos de datos, o dependencias entre ellos [2].

Esta extracción de conocimiento se consigue mediante técnicas de minería de datos. Con ellos obtenemos modelos de conocimiento a partir de los datos para poder realizar un análisis predictivo. Normalmente nos encontramos con dos tipos de problemas a resolver, la clasificación y la regresión.

- **Regresión:** En los problemas de regresión se trata de observar cómo reacciona una variable de respuesta (variable dependiente) en función de una variable explicativa (variable independiente).
- **Clasificación:** En los problemas de clasificación se trata de asignar una clase a un elemento entrante, el cual no tiene una categoría asignada.

Para la resolución de los problemas de clasificación, como el problema que está siendo tratado en este trabajo (donde tenemos las clases *1*, *X*, *2*), se utilizan los clasificadores. El término clasificador se utiliza en referencia al algoritmo utilizado para asignar a un elemento entrante no etiquetado, en una categoría concreta conocida. Dicho algoritmo permite ordenar o disponer por clases un conjunto de datos de entrada a partir de cierta información característica de éstos.

Una forma de implementar un clasificador es seleccionar un conjunto de datos de entrenamiento y tratar de definir unas ciertas reglas que permitan asignar una clase a cualquier otro dato de entrada. En ocasiones, el término clasificador también es utilizado para referirse a la función matemática que implementa el algoritmo de clasificación. En nuestro caso se van a utilizar varios métodos de clasificación para resolver el problema planteado. Dos de ellos son lo que se conoce como clasificadores débiles (weak learners) y el resto son métodos multiclasificadores. Los clasificadores débiles que va a usare son dos de los más utilizados en el ámbito del aprendizaje automático: los árboles de decisión y las redes neuronales artificiales.

3.1. Arboles de Decisión

Un árbol de decisión (en la literatura inglesa, *decision tree*) es un clasificador que lleva a cabo la partición recursiva sobre el espacio de instancias. Un árbol de decisión típico se compone de nodos internos y nodos hoja. Cada nodo interno representa una decisión sobre un atributo o un subconjunto de los atributos. De esta manera, los nodos internos dividen el espacio de instancias en dos o más particiones. Cada nodo hoja es un nodo terminal del árbol con una etiqueta de clase.

Debajo podemos ver en la Figura 5, un árbol de decisión básico, donde los nodos intermedios están representados por rectángulos, y los nodos hoja están representados con las clases Sí/No. En este ejemplo, tenemos tres atributos de división, "*Pronóstico*", "*Humedad*" y "*Viento*", y dos etiquetas de clase, "*Sí"* y "*No"*. Cada camino desde el nodo raíz al nodo hoja forma una regla de asociación.

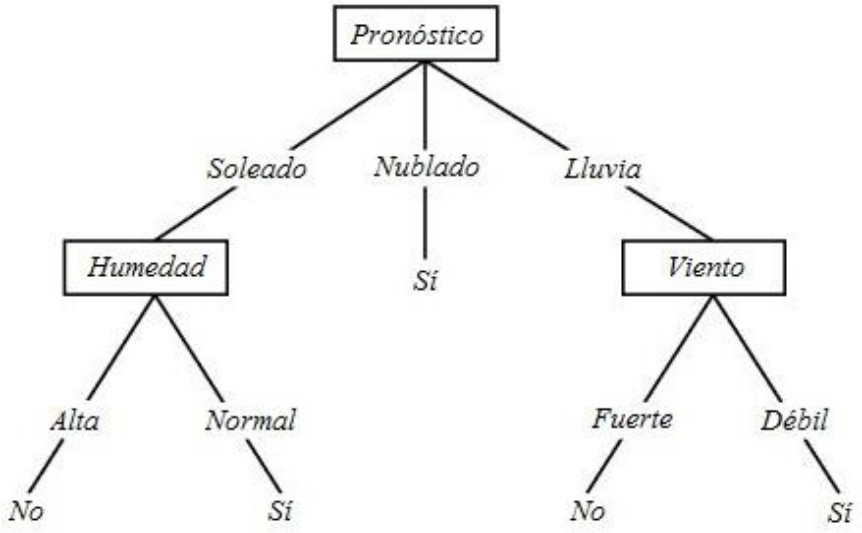

Figura 5. Ejemplo de árbol de decisión

El proceso general para la construcción de un árbol de decisión es el siguiente. Dado un conjunto de datos de entrenamiento, se aplica una función de medición en todos los atributos para encontrar la mejor partición de los atributos. Una vez que se determina el atributo de división, el espacio de instancias se divide en varias particiones. Dentro de cada partición, si todas las instancias pertenecen a una sola clase, el algoritmo termina. De lo contrario, se llevará a cabo de forma recursiva el proceso de división hasta que todas las particiones se asignen a una clase. Una vez construido el árbol de decisión, se pueden generar fácilmente las reglas de asociación, que se pueden utilizar para clasificar nuevas instancias con etiquetas de clase desconocidas.

En el trabajo se utiliza el algoritmo C4.5 [15], también conocido como J48. C4.5 es un algoritmo estándar para la inducción de reglas de asociación en forma de árbol de decisión. Fue desarrollado por Ross Quinlan en 1993, y representa una extensión del algoritmo ID3, desarrollado también por Quinlan. En C4.5, como criterio por defecto para la elección de los atributos de división se usa el ratio de ganancia de información. Este ratio evita el sesgo de selección de atributos con muchos valores.

C4.5 genera árboles de decisión desde un conjunto de datos de entrenamiento del mismo modo en que lo hace ID3. Posteriormente, en cada nodo del árbol, C4.5 elige un atributo de los datos que divide el conjunto de muestras en subconjuntos de forma más eficaz. El atributo con la mayor ganancia de información normalizada se elige como parámetro de decisión. El algoritmo C4.5 va pues dividiendo recursivamente el espacio de instancias en subconjuntos más pequeños.

3.2. Redes Neuronales

En el aprendizaje automático y la ciencia cognitiva, las redes neuronales artificiales (*Artificial Neural Network, ANN*) [16] son una familia de modelos de aprendizaje estadísticos inspirados en los sistemas nerviosos biológicos y cómo estos procesan la información. Se utilizan para estimar o aproximar funciones, generalmente desconocidas, que pueden depender de un gran número de entradas. Las redes neuronales artificiales se presentan generalmente como sistemas de elementos de procesamiento altamente interconectados (neuronas) que intercambian información entre sí para resolver problemas específicos. Cada una de estas conexiones tiene un peso que se puede ajustar en base a la experiencia, haciendo que las redes neurales sean adaptativas y capaces de aprender.

La estructura más común de red neuronal es el perceptrón multicapa. A diferencia del perceptrón simple (la red neuronal más simple), su estructura formada por múltiples capas, hace que pueda resolver problemas que no sean linealmente separables. El perceptrón multicapa puede ser totalmente conectado, cada salida de una neurona de la capa "i" es entrada de todas las neuronas de la capa "i+1" (ejemplo la Figura 6), o localmente conectado en caso contrario. Hay tres tipos de capas:

- Capa de entrada: neuronas que introducen en la red los patrones de entrada.
- Capas ocultas: formada por aquellas neuronas cuyas entradas provienen de capas anteriores y cuyas salidas pasan a neuronas de capas posteriores.
- Capa de salida: neuronas que devuelven las salidas de la red.

Como algoritmo de entrenamiento se suele utilizar la regla de retropropagación del error, también conocida como regla delta generalizada. Por ello, el perceptrón multicapa también es conocido como red de retropropagación.

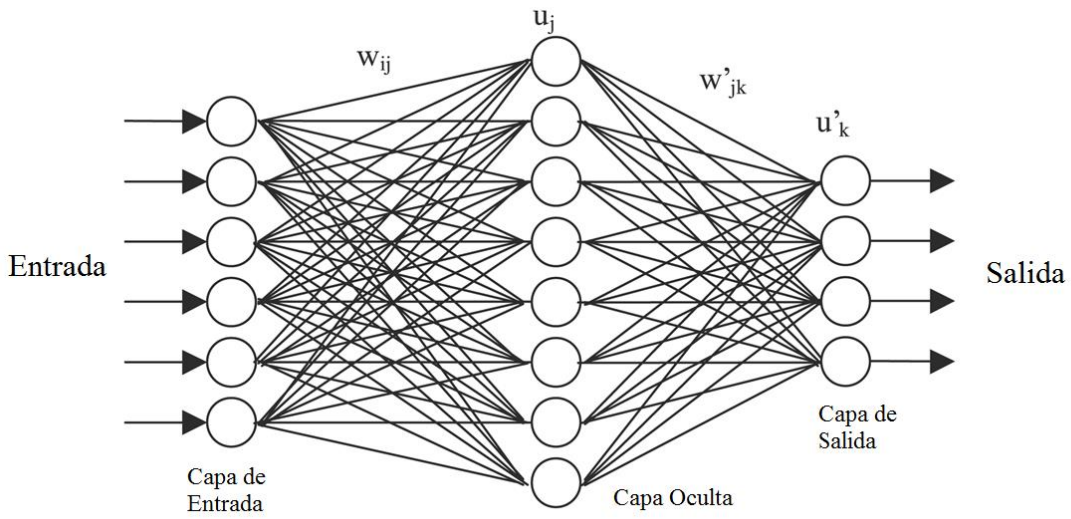

Figura 6. Red neuronal artificial

Las ventajas de las redes neuronales incluyen:

- **Patrones en los datos:** con su notable capacidad para entender el significado de datos complejos o imprecisos, las redes neuronales pueden extraer patrones y detectar tendencias que son demasiado complejas para ser observadas directamente por los seres humanos o a través de otras técnicas.

- **Aprendizaje adaptativo:** La capacidad de aprender a hacer tareas basadas en los datos dados para el entrenamiento o la experiencia inicial.

- **Auto-organización:** Una ANN puede crear su propia organización o representación de la información que recibe durante el tiempo de aprendizaje.

- **Operaciones en tiempo real:** la computación de los ANN pueden llevarse a cabo en paralelo, y el diseño de dispositivos de hardware especializados están explotando esta capacidad.

- **Tolerancia a fallos a través de la codificación de información redundante:** la destrucción parcial de una red conduce a la degradación correspondiente de rendimiento. Sin embargo, algunas habilidades de la red pueden ser retenidas incluso con daños a la red principal.

3.3. Multiclasificadores

El uso de los sistemas multiclasificadores se ha visto incrementado en ciertos ámbitos de investigación, en detrimento de los clasificadores tradicionales, gracias a que resuelven ciertos problemas de sobreadaptación (overfitting) [5]. Esto es debido a que los multiclasificadores tienen en cuenta, en mayor o menor medida, todas las hipótesis válidas, realizando una combinación de las diferentes predicciones obtenidas. Así mismo, gracias a la combinación de los distintos clasificadores individuales, es posible descomponer un problema complejo en sub-problemas los cuáles puedan ser más fáciles de resolver y entender. Por medio de la combinación incluso pueden eliminarse los errores no-correlacionados de los mencionados clasificadores individuales.

Los multiclasificadores son, por lo tanto, conjuntos de clasificadores diferentes que realizan predicciones que se fusionan y se obtiene como resultado la combinación de cada una de ellas [3]. La idea de combinación hace que en la literatura se cite a los multiclasificadores a través de distintos términos, tales como: métodos de *ensamble*; modelos múltiples (*multiple models*); sistemas de múltiples clasificadores (*multiple classifier systems*); combinación de clasificadores (*combining classifiers*); mezcla de expertos (*mixture of experts*); o comité de expertos (*committee of experts*).

A la hora de distinguir unos multiclasificadores de otros, se puede atender a diversas características. Así, podemos distinguirlos por el número de clasificadores individuales que se integran, el tipo de cada clasificador individual (árboles de decisión, redes neuronales, etc.), la combinación de las decisiones particulares (voto mayoritario, asignación de pesos, medidas de promedio, etc.), o la naturaleza o el tamaño de los conjuntos de datos de entrenamiento usados en cada clasificador.

Para evaluar los distintos multiclasificadores se pueden establecer diferentes medidas de rendimiento, que incluyen el grado de generalización, el número de clasificaciones correctas e incorrectas, y el tiempo real de ejecución.

Para este trabajo, han sido utilizado tres de los multiclasificadores con más literatura y con un mayor rendimiento a la hora de trabajar con distintos problemas de clasificación. Estos son, el método de Random Forest, el método Bagging y el algoritmo AdaBoost.M1. De ellos se hablará más adelante, ahora vamos a ver las dos características más importantes a la hora de la creación de un multiclasificador: la arquitectura que presenta y el método de combinación que utiliza.

3.3.1. Arquitectura

La arquitectura de un multiclasificador es la forma en la que este integra al conjunto de clasificadores individuales de los que se compone. Según el comportamiento que se desee que tengan estos, se tienen que configurar en unas arquitecturas u otras. Por ejemplo, se pueden integrar para conseguir una cooperación para la resolución del problema, o para que cada uno de ellos intente generar su propia hipótesis. De acuerdo a ello los multiclasificadores pueden adoptar distintas arquitecturas:

- **Arquitectura en serie o vertical:**
  En este modelo los clasificadores individuales están conectados unos a otros, de forma que, la información resultante de un clasificador se convierte en el conjunto de datos de entrada del siguiente, tal como se ve en la Figura 7.

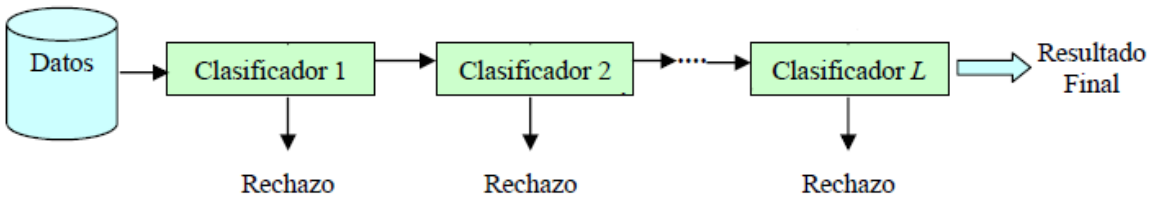

Figura 7. Arquitectura en serie o vertical

  Cuando se adopta esta arquitectura, se obtiene la ventaja de un filtrado progresivo de las decisiones. Sin embargo, es un modelo sensible al orden, por lo que se debe tener un conocimiento a priori del comportamiento de cada clasificador. Además, de manera general, es difícil de optimizar el conjunto de clasificadores ya que suele existir una dependencia entre ellos.

- **Arquitectura en paralelo u horizontal:**
  En esta arquitectura, los clasificadores son integrados para operar independientemente unos de otros, a diferencia de la arquitectura vertical. La información resultante de cada uno de ellos es fusionada, buscando un consenso para llegar a una única decisión.

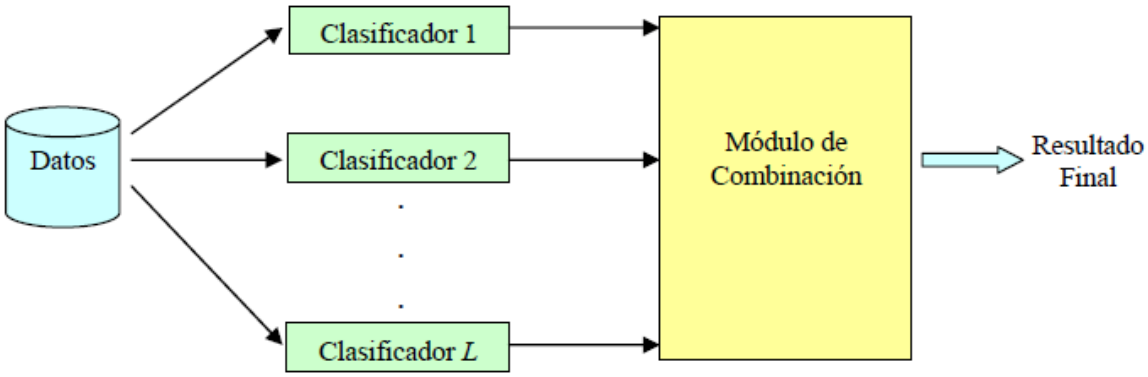

Figura 8. Arquitectura en paralelo u horizontal

Este modelo es fácil de aplicar, ya que no requiere reparametrizar el conjunto de datos, y puede integrar clasificadores de diversa índole. Su desventaja radica en un costoso tiempo de cálculo debido a la activación simultánea de todos los clasificadores.

- **Arquitectura híbrida:**
  La arquitectura híbrida es una combinación de las arquitecturas en serie y paralela, por lo que también combina sus ventajas. Este modelo presenta pues una reducción del conjunto de las clases posibles, así como la búsqueda de un consenso entre los clasificadores.

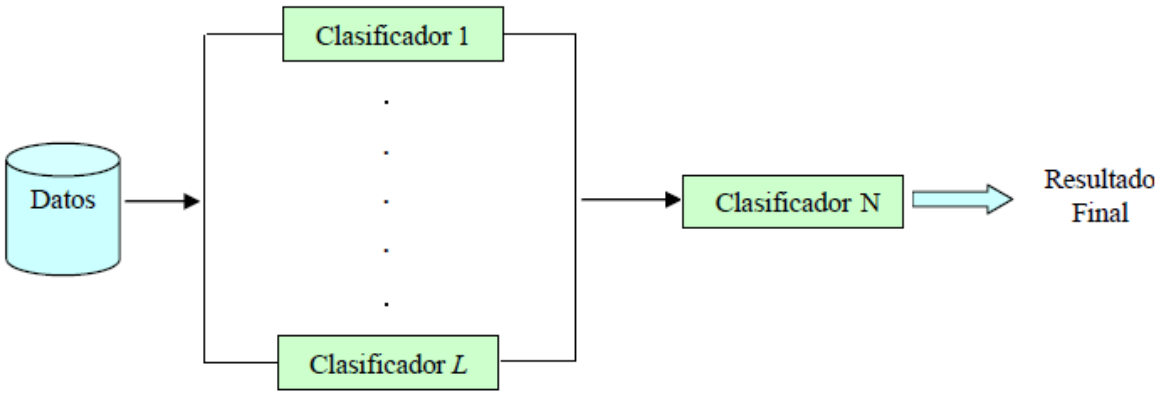

Figura 9. Arquitectura híbrida

Uniendo ambas arquitecturas, se puede obtener un mejor provecho de cada uno de los clasificadores que se integran, pero los datos se tratan de forma dependiente, por lo que al igual que en la arquitectura vertical, es más complejo de optimizar que si se tuviera un esquema en paralelo.

3.3.2. Métodos de combinación

Como ya se ha comentado, una vez obtenidas las salidas de los clasificadores individuales, estas deben ser fusionadas para obtener la respuesta del conjunto. Para ello disponemos de diferentes métodos de combinación, y aunque sabemos que las salidas de los clasificadores pueden tomar diferentes formatos, la filosofía de la mayoría de métodos se puede aplicar a cualquier modelo.

- **Voto mayoritario simple y Voto mayoritario por peso:**
  La técnica de voto mayoritario simple es la técnica más común de votación para el ser humano. La salida de cada clasificador individual es un voto, y cada voto tiene el mismo valor. En definitiva, la clase más votada es la que sale elegida por el conjunto como decisión final.

  El voto mayoritario por peso es similar al anterior, pero con la diferencia de que cada voto tiene un valor distinto según el clasificador que lo haya emitido. De esta forma, el resultado emitido por el conjunto es el voto que, tras aplicarle los pesos, resulte mayoritario.

  Un enfoque para el uso de esta técnica consiste en entrenar a cada clasificador individualmente, y evaluar a cada uno con unos datos de prueba fijados. Entonces se le asigna un peso a cada voto según la evaluación realizada. Si se asume que estos resultados obtenidos en la prueba representan la eficacia de cada clasificador individual, es más probable que este método de combinación proporcione mejores resultados que el del voto mayoritario simple.

  La ventaja de estos métodos es que pueden ser aplicados para cualquier combinación de clasificadores, aunque se impone una alta demanda en el tamaño y la calidad del conjunto de datos.

- **Promedio simple y otras medidas estadísticas:**
  Una de los métodos de combinación más simples se basa en promediar las salidas de los clasificadores individuales. Se puede definir un vector de probabilidades *v(k)* donde *k* es el número de clases del problema. Si se tienen *m* clasificadores, se unifican en único vector α, el cual será utilizado para extraer la clase predicha, que será la clase que obtenga la mayor probabilidad. El promedio será calculado entonces como la suma de las probabilidades de una misma clase en todos los clasificadores dividido entre el número de éstos:

$$\alpha = \sum_{j=1}^{m} \frac{v_j}{m}$$

Otros operadores estadísticos son por ejemplo, la suma, la mediana, o el máximo y el mínimo.

- Para calcular la suma, se suman las probabilidades de una misma clase en todos los clasificadores.
- Para calcular la mediana se ordenan de menor a mayor los valores de las probabilidades de una misma clase en todos los clasificadores. Si tenemos un número de clasificadores impar, la mediana es el valor del individuo que ocupa el valor central, y si tenemos que es par, la mediana es el valor medio de los datos centrales.
- Para el máximo se busca el valor superior de las probabilidades de una misma clase en todos los clasificadores. En el caso del mínimo se aplica la misma regla pero buscando el valor inferior.

Tenemos entonces que:

$$\text{Suma:} \quad \alpha = \sum_{j=1}^{m} v_j$$

$$\text{Mediana:} \quad \alpha = mediana_{i \le j < m}(v_j)$$
$$\text{Máximo:} \quad \alpha = máximo_{i \le j < m}(v_j)$$
$$\text{Mínimo:} \quad \alpha = mínimo_{i \le j < m}(v_j)$$

3.4. Random Forest

Random Forest es un método de aprendizaje que se puede usar para clasificación, regresión u otras tareas. El algoritmo para creación de un random forest fue desarrollado por Leo Breiman y Adele Cutler [9]. El método combina la idea de Breiman de *"bagging"* y la selección aleatoria de características, introducida de forma independiente por Ho, Amit y Geman, con el fin de construir una colección de árboles de decisión con una varianza controlada.

Random Forest opera mediante la creación de muchos árboles de decisión (que forman un bosque). Cada árbol es construido usando el siguiente algoritmo:

1. Se elige un conjunto de datos de entrenamiento para entrenar el árbol y se usa el resto de los datos como datos de prueba para estimar el error cometido.
2. Siendo *N* el número de casos en el conjunto de entrenamiento y *M* el número de atributos del conjunto de entrada.
3. Siendo *m* el número de atributos que van a ser usados para determinar la decisión en un nodo intermedio, y siendo $m << M$.
4. Para cada nodo del árbol de decisión, elegir aleatoriamente *m* atributos sobre los cuales se tomará la decisión. Calcular la mejor partición a partir de los *m* atributos escogidos.

A la hora de la predicción de una nueva instancia, esta pasa como entrada de todos los árboles de decisión que componen el bosque. Cada uno de los árboles devuelve una predicción sobre la clase a la que pertenece la nueva instancia. Una vez todos

han devuelto una clase, el bosque toma como predicción correcta aquella que tenga mayor número de incidencias en las respuestas de los árboles individuales.

El método de random forest también se usa para medir la importancia de los atributos de un problema de regresión o de clasificación. Para medir la importancia del atributo enésimo, una vez concluido el entrenamiento, se permutan los valores de este atributo y se calcula el error con el conjunto de datos permutado. Se promedia la diferencia en el error antes y después de la permutación de todos los árboles, y se normaliza por la desviación estándar de estas diferencias, obteniendo una puntuación para el atributo. Los atributos con grandes valores de puntuación se clasifican como más importantes que los atributos con valores pequeños.

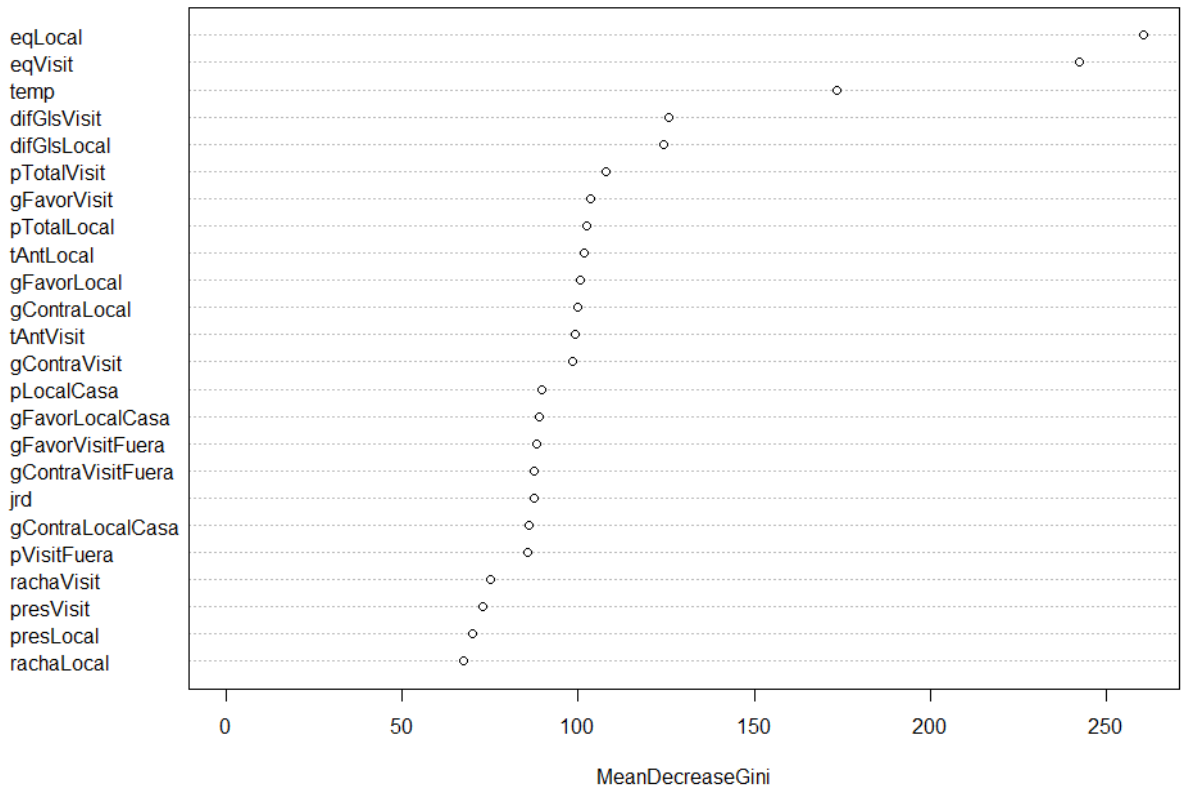

Figura 10. Importancia Gini obtenida de Random Forest

En la Figura 10 se está representando la importancia de los atributos que utilizamos en nuestro problema. Estos resultados han sido obtenidos gracias a una implementación de los random forest con el paquete R "*randomForest*" [7].
Se pueden sacar varias conclusiones de la figura anterior. Entre ellas:

- La mayor importancia recae sobre cuáles son los equipos que juegan el encuentro, siendo más relevante cuál es el equipo que juega como local.

- Lo siguiente más relevante es en qué temporada se juega el partido, ya que muchos equipos están más en forma o tienen mejores jugadores en unas temporadas que en otras.

- La diferencia de goles es más importante que los propios goles a favor o en contra, o que los puntos que lleva el equipo. Esto demuestra que es un buen indicativo del estado de un equipo.

- Atributos que en principio pueden parecer más importantes a la hora de predecir el resultado, como son el presupuesto de los equipos o la racha que llevan, no son demasiado relevantes para este propósito, si bien su ausencia puede hacer que perdamos poder a la hora de predecir.

- La media de las importancias de los atributos correspondientes al equipo local es levemente superior a la media de las importancias de los atributos del equipo visitante. Esto nos indica que, sabiendo cuál es el equipo local, también es relevante el estado en el cual el equipo visitante llega al encuentro.

3.5. Bagging

Bagging es una técnica de aprendizaje automático propuesta por Leo Breiman [6]. Esta técnica es utilizada para generar múltiples versiones de un mismo clasificador e integrarlos para obtener una predicción conjunta. Para combinar los resultados de las distintas versiones, utiliza el método del voto de pluralidad, o método del voto mayoritario simple, cuando el objetivo a predecir es una clase; y el promedio cuando el objetivo es un resultado numérico.

El *bootstrap* [5] es una técnica que se basa la generación de conjuntos de datos de entrenamiento mediante la selección aleatoria uniforme de instancias y su duplicación con reemplazo. De esta forma, se crean tantos conjuntos mediante *bootstrap* como versiones del clasificador haya, y cada una de las versiones entrena con el conjunto que se le haya asignado.

Una aproximación al algoritmo bagging sería:

ENTRADA conjunto de entrenamiento *S*, inductor *I*, número de muestras *boostrap T,*
(el inductor es el método de clasificación, ej. árbol de decisión)

PARA i = 1

S' = muestra generada por boostrap a partir de *S*
Entrenamos el clasificador con la nueva muestra: $C_i = I(S')$

HASTA *T*

C(x) =* $\arg\max_{y \in Y} \sum_{i: C_i(x) = y} 1$ *(en este ej. Voto mayoritario simple)*

SALIDA clasificador final C*

Pruebas realizadas en diferentes conjuntos de datos reales y simulados, utilizando árboles de decisión y de regresión y regresiones lineales, muestran que el método bagging presenta una mejora sustancial de la precisión a la hora de predecir. El elemento esencial es la inestabilidad del método de predicción, es decir, si cambiando ligeramente el conjunto de entrenamiento, el clasificador individual presenta cambios significativos a la hora de predecir, significa que el clasificador es inestable. En estos casos, el uso de bagging con este clasificador, mejora la precisión que presenta el mismo. Aunque se puede aplicar con cualquier clasificador débil (*weak learners*), se aplica generalmente con árboles de decisión o de regresión.

3.6. AdaBoost.M1

AdaBoost, abreviatura de "*Adaptive Boosting*", es un algoritmo de aprendizaje automático diseñado por Yoav Freund y Robert Schapire [8], y los cuales ganaron el prestigioso "Premio Gödel" en 2003 por su trabajo. Al igual que ocurre con las técnicas de Boosting y Bagging, puede aplicarse con diferentes tipos de algoritmos de aprendizaje para mejorar el rendimiento de los mismos.

Las salidas de los clasificadores débiles (*weak learners*) se combinan en una suma ponderada, la cual representa la salida final del conjunto. AdaBoost es adaptativo en el sentido de que posteriores clasificadores se ajustan según casos mal clasificados por clasificadores anteriores. AdaBoost es sensible a los datos ruidosos y valores atípicos, sin embargo, en algunos problemas puede ser menos susceptible al problema del sobreajuste (overfitting) que otros algoritmos de aprendizaje. Los clasificadores individuales aplicados pueden ser de cualquier naturaleza, siempre y cuando su rendimiento sea ligeramente mejor que el azar (es decir, su tasa de error es menor que 0,5 para una clasificación binaria).

Mientras que cada algoritmo de aprendizaje tiende a adaptarse a algunos tipos de problemas mejor que otros, y suelen tener muchos parámetros y configuraciones diferentes para ajustarse antes de conseguir un rendimiento óptimo con un conjunto de datos, AdaBoost (con árboles de decisión como clasificadores individuales) suele obtener un buen resultado sin demasiada carga de trabajo. Cuando se aplica con árboles de decisión, la información recogida en cada etapa del algoritmo AdaBoost acerca de la 'dureza' relativa de cada muestra del entrenamiento, se le pasa al algoritmo de creación del árbol, de tal manera que los árboles que se creen posteriormente tiendan a centrarse en los ejemplos más difíciles de clasificar.

AdaBoost.M1 [8] es una extensión de este algoritmo AdaBoost. El funcionamiento es prácticamente el mismo, si bien, AdaBoost fue diseñado para problemas en los cuales se tuviera solo 2 clases, es decir, solo resuelve problemas binarios. En cambio, gracias a esta extensión, se puede utilizar este modelo para la resolución de problemas multiclase, de ahí la extensión M.

El algoritmo AdaBoost.M1 [7] funciona tal que:

ENTRADA conjunto de entrenamiento *S = {(x1; y1),…, (xi; yi),… (xn; yn)}*
donde *yi* es la clase y toma valores en *1, 2,…,k.*

INICIALIZAR los pesos $w_t(i)$ = 1/n; i = 1, 2,… n

PARA t = 1

Entrenar el clasificador $C_t(x_i) = \{1, 2, \ldots k\}$ usando pesos $w_t(i)$ en S_t

Calcular el error $e_t = \sum_{i=1}^{n} w_b(i)\, \boldsymbol{I}(C_t(x_i) \neq y_i)$

Calcular el ratio de error $\alpha_t = 1/2 \ \ln\ ((1 - e_t)/e_t)$

Actualizar pesos $w_{b+1}(i) = w_b(i) \exp(\alpha_t\, \boldsymbol{I}(C_t(x_i) \neq y_i))$ y normalizarlos

HASTA *T*

$C^*(x_i) = \arg\max_{y \in Y} \sum_{t=1}^{T} \alpha_t\, \boldsymbol{I}(C_t(x_i) = y)$

SALIDA clasificador final $C^*(x_i)$

Donde $\mathbf{I}$ es una función tal que: $\boldsymbol{I}(x) = \begin{cases} 1 & si\ \ x = True \\ 0 & si\ x = False \end{cases}$

La propiedad más importante sobre AdaBoost.M1 se indica en el siguiente teorema: "*Si las hipótesis tienen consistentemente un error ligeramente mejor que un medio, entonces, el error de la hipótesis final cae a cero rápidamente de forma exponencial*". Para problemas de clasificación binaria, esto significa que las hipótesis tienen por qué ser sólo ligeramente mejor que el azar para que la hipótesis final sea la mejor.

Este teorema apoya la idea principal de que con los multiclasificadores se obtienen mejores resultados que los mejores que se pueden obtener con los clasificadores individuales que los integran [4]. Si bien, esta idea no es un teorema, por lo que no se puede aplicar a todos los multiclasificadores que se desarrollan, como veremos más adelante en el apartado de análisis. Sin embargo si se cumple para aquellos que tienen una amplia base matemática detrás, como son todos los métodos basados en boosting y bagging con la integración de clasificadores débiles.

4. Implementación de los multiclasificadores desarrollados

En este apartado van a ser presentados los diferentes algoritmos de aprendizaje automático diseñados por nosotros. Todos estos modelos, o métodos, son diseñados como métodos multiclasificadores, y presentan dos tipos de estructuración. De este modo, nos encontramos con que algunos de ellos presentan una arquitectura paralela con un método de combinación para obtener el resultado final, y el resto presentan una arquitectura híbrida de dos niveles conectados en serie.

En ambos diseños se tiene un nivel de clasificadores individuales integrados en paralelo. En la arquitectura híbrida, este será el primer nivel, al cual se le pasan las entradas del problema. Los clasificadores individuales elegidos para esta integración en paralelo son las redes neuronales artificiales y los árboles de decisión. Ambos han sido explicados anteriormente en el trabajo y son dos de los clasificadores débiles más utilizados en el aprendizaje automático. En el diseño con la arquitectura en paralelo, los resultados obtenidos de estos clasificadores serán fusionados mediante un método de combinación para obtener un resultado final. En el diseño con arquitectura híbrida, estos resultados serán recogidos en un dataframe intermedio.

En el segundo nivel de la arquitectura híbrida, tendremos un solo clasificador simple trabajando en serie con los anteriores. Este recibirá de ellos el dataframe intermedio anteriormente mencionado, el cual pasara a ser su conjunto de datos de entrada. Este clasificador será o bien una red neuronal, o bien un árbol de decisión, y será del que, en última instancia, obtendremos los resultados de la predicción.

A la hora de la implementación de estos modelos en el lenguaje R, hemos separado el entrenamiento de la predicción. De esta forma lo que tenemos es un programa R que crea la función de entrenamiento, a la cual se debe pasar como parámetros el conjunto de datos de entrenamiento, y que nos devuelve un resumen de los resultados obtenidos tras la predicción de los propios datos de entrenamiento; y un segundo programa R que crea la función de predicción, a la que le pasamos como parámetro el conjunto de datos de prueba y que nos devuelve un resumen de los resultados obtenidos tras la predicción.

Para su seguimiento y diferenciación, se le ha otorgado un nombre en clave a cada modelo, de forma que todos ellos empiezan por las siglas "*TFG*" y continúan con un número identificativo. Las variantes que nos encontramos entre modelos del mismo tipo pueden ser cambios en el número de clasificadores, cambios de estructura, etc.

- **0:** implementaciones de clasificadores y multiclasificadores de terceros.
- **1 – 19:** variantes del diseño con arquitectura híbrida.
- **20 – 29:** variantes de arquitectura paralela con votación mayoritaria simple.
- **30 – 39:** variantes de arquitectura paralela con votación mayoritaria por pesos.
- **40:** variante de arquitectura en paralela con cambio en los datos.

Para las implementaciones se utiliza el lenguaje de programación para análisis estadístico y gráfico, R. R proporciona un amplio abanico de herramientas estadísticas (como algoritmos de clasificación y agrupamiento), y una fácil extensión a través de paquetes desarrollados por su amplia comunidad de usuarios.

Además, hay desarrollada una interfaz en R, llamada *RWeka* [18] para interactuar con el conocido software para el aprendizaje automático y la minería de datos, Weka. Gracias a esta interfaz podemos usar en el lenguaje R los algoritmos de minería de datos de dicha plataforma.

4.1. Diseños externos

- **TFG 0; Árbol de decisión J48**

J48, también conocido como C4.5, fue desarrollado por Ross Quinlan (1993) [15]. La implementación de este algoritmo que se utiliza en el trabajo, es la ofrecida en el paquete R "*RWeka*". La forma de uso es:

J48(formula, data, subset, na.action, control = Weka_control(), options = NULL)

Para nuestra implementación hemos usado:

- **Formula:** quiniela ~. Esto significa que el atributo a clasificar es "*quiniela*" y que para el entrenamiento del árbol de decisión, vamos a hacer uso del resto de atributos que presenten los datos.
- **Data:** dtrain. Conjunto de datos de entrenamiento.
- **Control:** son parámetros de control de los clasificadores de Weka. En nuestro caso necesitamos usar:
    - **Parámetro de control U:** indica el método de poda con el cual se quiere crear el árbol. De modo que, si se le indica TRUE, significa que se quiere un árbol de decisión pre-poda; y si se indica FALSE o no se indica nada, significa que queremos crear el árbol mediante post-poda.
    - **Parámetro de control M:** indica el mínimo número de instancias por nodo que queremos, es decir, debe haber al menos M instancias cuya predicción venga dada por la terminación en ese nodo.

Para obtener la predicción resultante de un árbol de predicción, se puede utilizar la función de predicción que nos proporciona R. En ella, hay que indicar el método ya entrenado, y el conjunto de datos a predecir. Ejemplo: *predict(decTree, dtest)*

Al igual que vamos a hacer con las implementaciones de los modelos creados por nosotros, hay que diferenciar en un programa R en el cual se hace el entrenamiento, y otro en el que se hace la predicción con el conjunto de datos de prueba.

```
Entrenamiento <- function(dtrain) {

  library(RWeka)
  nrows <- nrow(dtrain)

  decTree <<- J48(quiniela ~ ., data = dtrain, control = Weka_control(U =
TRUE, M = 20))

  decTreePred <- predict(decTree, newdata = dtrain)
  t <- table(decTreePred, dtrain$quiniela)
  Visualizacion(t,nrows)
}
```

```
Prediccion <- function(dtest) {

  library(RWeka)
  nrows <- nrow(dtest)

  decTreePred <- predict(decTree, newdata = dtest)
  t <- table(decTreePred, dtest$quiniela)
  Visualizacion(t,nrows)
}
```

- **TFG 0; Red neuronal artificial ANN**

Gracias al paquete R "*nnet*" [19], se pueden usar redes neuronales artificiales, si bien presenta la restricción de que estas solamente tienen una capa oculta de neuronas. La implementación de estas redes se basa en la desarrollada por Brian Ripley [16]. La forma de uso es:

nnet(formula, data, weights, size, maxit, subset, na.action, decay, trace, …, contrasts)

Para nuestra implementación hemos usado:

- **Formula:** quiniela ~. El atributo a clasificar es "*quiniela*" y para su clasificación, vamos a hacer uso del resto de atributos que presenten los datos.
- **Data:** dtrain. Conjunto de datos de entrenamiento.
- **Size:** es el número de neuronas que se quiere en la capa oculta.
- **Maxit:** número máximo de iteraciones que se quieren realizar.
- **Decay:** 1. Establece los "*weight decay*" (pesos decadentes). Es una medida para limitar el efecto de los pesos altos.
- **Trace:** FALSE. De este modo evitamos que R nos presente todas las iteraciones que son necesarias para llegar a la convergencia del modelo.

```
Entrenamiento <- function(dtrain) {

  library(nnet)
  nrows <- nrow(dtrain)
  preds.final <- matrix(0,nrows,1)

  # ---------------------- Red Neuronal (7) -----------------------------

  nn.fit <<- nnet(quiniela~., data=dtrain, size=7, maxit=500, decay=1,
trace=FALSE)
  nn.pred <- predict(nn.fit,dtrain,type="raw")

  for(j in 1:nrows) {
    pred1 <- nn.pred[j,1]
    predX <- nn.pred[j,3]
    pred2 <- nn.pred[j,2]

    if(pred1 > predX && pred1 > pred2){       preds.final[j,1] <- "1"
    }else if(predX > pred1 && predX > pred2){ preds.final[j,1] <- "X"
    }else if(pred2 > pred1 && pred2 > predX){ preds.final[j,1] <- "2" }
  }

  t <- table(preds.final,dtrain$quiniela)
  Visualizacion(t,nrows)
}
```

```
Prediccion <- function(dtest) {

  library(nnet)
  nrows <- nrow(dtest)
  preds.final <- matrix(0,nrows,1)

  # ---------------------- Red Neuronal (7) -----------------------------

  nn.pred <- predict(nn.fit,dtest,type="raw")

  for(j in 1:nrows) {
    pred1 <- nn.pred[j,1]
    predX <- nn.pred[j,3]
    pred2 <- nn.pred[j,2]

    if(pred1 > predX && pred1 > pred2){       preds.final[j,1] <- "1"
    }else if(predX > pred1 && predX > pred2){ preds.final[j,1] <- "X"
    }else if(pred2 > pred1 && pred2 > predX){ preds.final[j,1] <- "2" }
  }

  t <- table(preds.final,dtest$quiniela)
  Visualizacion(t,nrows)
}
```

Cuando se hace una predicción sobre una red neuronal, con la función *predict*, lo que esta devuelve son las probabilidades de que la entrada pertenezca a cada una de las clases en las que pueda ser clasificada. Por esta razón, si queremos saber en qué clase debe ser clasificada la entrada, tenemos que buscar la clase que tenga una probabilidad mayor devuelta por la red. Entonces, para un conjunto de datos, lo que hacemos es recorrer todas las respuestas de la red, y crear una estructura de datos que recoja la clase que tenga mayor probabilidad para cada una de ellas.

- **TFG 0; Método AdaBoost.M1**

Esta implementación está basada en el algoritmo AdaBoost M1 de Freund y Schapire (1996) [8]. La forma de uso es:

AdaBoostM1(formula, data, subset, na.action, control = Weka_control(), options)

Para nuestra implementación hemos usado:

- **Formula:** quiniela ~. El atributo a clasificar es “quiniela” y que para ello vamos a hacer uso del resto de atributos que presenten los datos.
- **Data:** dtrain. Conjunto de datos de entrenamiento.
- **Control:** Se va a utilizar una lista de árboles de decisión J48 con profundidad máxima de 50 niveles.

```
Entrenamiento <- function(dtrain) {

  library(RWeka)
  nrows <- nrow(dtrain)

  adaboostm1 <<- AdaBoostM1(quiniela ~ ., data = dtrain, control =
Weka_control(W = list(J48, M = 50)))

  predm1 <- predict(adaboostm1, newdata = dtrain)
  t <- table(predm1,dtrain$quiniela)
  Visualizacion(t,nrows)
}
```

```
Prediccion <- function(dtest) {

  library(RWeka)
  nrows <- nrow(dtest)

  predm1 <- predict(adaboostm1, newdata = dtest)
  t <- table(predm1, dtest$quiniela)
  Visualizacion(t,nrows)
}
```

- **TFG 0; Método Bagging**

Esta implementación nos provee del método bagging (Breiman, 1996) [6]. La forma de uso es similar a la que hemos visto en la implementación de AdaBoost M1:

Bagging(formula, data, subset, na.action, control = Weka_control(), options)

Para nuestra implementación hemos usado:

- **Formula:** quiniela ~. El atributo a clasificar es “quiniela” y que para ello vamos a hacer uso del resto de atributos que presenten los datos.
- **Data:** dtrain. Conjunto de datos de entrenamiento.
- **Control:** Se va a utilizar una lista de árboles de decisión J48 con profundidad máxima de 30 niveles.

```
Entrenamiento <- function(dtrain) {

  library(RWeka)
  nrows <- nrow(dtrain)

  bagging <<- Bagging(quiniela ~ ., data = dtrain, control = Weka_control(
W = list(J48, M = 30)))

  predm1 <- predict(bagging, newdata = dtrain)
  t <- table(predm1,dtrain$quiniela)
  Visualizacion(t,nrows)
}
```

```
Prediccion <- function(dtest) {

  library(RWeka)
  nrows <- nrow(dtest)

  predm1 <- predict(bagging, newdata = dtest)
  t <- table(predm1, dtest$quiniela)
  Visualizacion(t,nrows)
}
```

En todas las implementaciones, una vez obtenidos los resultados, se imprime por pantalla un resumen de estos. Usamos una función, desarrollada por nosotros mismos, la cual nos dice las instancias correcta e incorrectamente clasificadas, además de sus porcentajes y la matriz de confusión obtenida. En la mayoría de clasificadores, esto puede ser obtenido usando la función *summary()*, sin embargo, no es así en las redes neuronales, por ellos decidimos hacer nuestra propia función.

4.2. Diseños propios

- **TFG 1; Paralelo: 1 red neuronal y 1 J48; Serie: 1 J48**

Es el diseño más básico del sistema.

En el entrenamiento se toman los datos del conjunto de entrenamiento y se les pasa a una red neuronal y a un árbol de decisión J48. Para poder formar el dataframe intermedio se cogen las predicciones directamente del árbol de decisión, pero para obtener las de la red neuronal se ha de hacer primero una predicción de la red sobre los propios datos de entrenamiento debido a la implementación de este clasificador simple en el paquete R "*nnet*" [19]. Esta predicción viene dada en forma de probabilidades de las clases de los datos, por lo que posteriormente a la predicción ha de obtenerse una decisión comparando las probabilidades y escogiendo la clase de mayor probabilidad.

Una vez construido el dataframe, se concluye el entrenamiento en paralelo. El dataframe construido se pasa al árbol de decisión J48 dispuesto en serie, pasando a ser sus datos de entrenamiento. Una vez entrenado, se devuelven los resultados.

```
Entrenamiento <- function(dtrain) {

  library(nnet)
  library(RWeka)

  nrows <- nrow(dtrain)
  preds <- matrix(0,nrows,1)

  # ---------------------- Red Neuronal (7) -----------------------------

  nn.fit <- nnet(quiniela~., data=dtrain, size=7, maxit=500, decay=1,
trace=FALSE)
  nn.pred <- predict(nn.fit,dtrain,type="raw")

  for(j in 1:nrows) {
    pred1 <- nn.pred[j,1]
    predX <- nn.pred[j,3]
    pred2 <- nn.pred[j,2]

    if(pred1 > predX && pred1 > pred2){       preds[j,1] <- "1"
    }else if(predX > pred1 && predX > pred2){ preds[j,1] <- "X"
    }else if(pred2 > pred1 && pred2 > predX){ preds[j,1] <- "2" }
  }

  # ---------------------- Arbol J48 ----------------------------------

  arbolJ48 <- J48(quiniela ~ ., data = dtrain, control = Weka_control(U =
TRUE, M = 20))
```

```
  # ---------------- Creacion del dataframe -----------------------------
  # --------- Finaliza el entrenamiento en paralelo ---------------------

  A <- data.frame(preds[,1], arbolJ48$predictions, dtrain[,ncol(dtrain)])
  names(A) <- c("RedNeur","Arbol","quiniela")

  # ------------- Arbol J48 dispuesto en serie --------------------------

  arbol.Final <<- J48(quiniela ~ ., data = A)
  final.predict <- predict(arbol.Final, newdata = A)
  t <- table(final.predict,A$quiniela)

  Visualizacion(t,nrows)
}
```

En este modelo, nos encontramos con una red neuronal multicapa con 7 neuronas en una sola capa oculta, ya que la implementación R no deja tener dos o más capas ocultas, y 3 neuronas en la capa de salida, además de un número máximo de iteraciones de 500 (*maxit*). En el caso del árbol de decisión, tenemos un J48 o C4.5, aunque el árbol creado para el primer y segundo nivel es distinto. Antes de continuar, expliquemos brevemente que es la poda.

La poda es una técnica de aprendizaje automático que se encarga de reducir el tamaño de los árboles de decisión eliminando secciones del mismo que aporten poco potencial a la hora de clasificar. De esta forma, lo que se pretende es evitar el sobreajuste (*overfitting*). El sobreajuste surge a medida que añadimos niveles al árbol de decisión y las hipótesis se van refinando tanto que describen muy bien los ejemplos que se utilizan en el aprendizaje, pero el error con los datos de prueba aumenta. Es debido a que se aprende el ruido del conjunto de entrenamiento, llegándose a adaptar a las regularidades del mismo y no generalizando al conjunto de datos de prueba. Tenemos dos técnicas de poda básicas:

- **Pre-poda:** son técnicas que tratan de detener el crecimiento del árbol antes de que el mismo llegue a formarse de forma que se adapte al conjunto de datos de entrenamiento.

- **Post-poda:** deja que el árbol se forme con sobreajuste sobre los datos y luego efectúa una poda sobre él eliminando ramas del mismo.

En este diseño tenemos pues dos árboles distintos, uno al que se le ha aplicado pre-poda, y otro creado mediante post-poda. Para el primer nivel se ha utilizado un árbol creado mediante pre-poda (*U = TRUE*), en el cual se le pone una restricción de mínimo 20 instancias clasificadas a través del nodo (*M = 20)*. El árbol del segundo nivel sin embargo, ha sido creado mediante post-poda.

Para la predicción se toman los datos del conjunto de prueba y se pasan a la red neuronal y al árbol de decisión J48. Al igual que en el entrenamiento, para formar el dataframe intermedio se cogen las predicciones directamente del árbol de decisión, y las de la red neuronal escogiendo la clase de mayor probabilidad. Ese dataframe se le pasa al árbol de decisión J48 dispuesto en paralelo para que haga la predicción.

Una vez hecho todo esto, se devuelve el resumen de los resultados obtenidos de la predicción del conjunto.

```
Prediccion <- function(dtest) {

  library(nnet)
  library(RWeka)

  nrows <- nrow(dtest)
  preds <- matrix(0,nrows,4)

  # ---------------------- Red Neuronal (7) -----------------------------

  nn.pred <- predict(nn.fit,dtest,type="raw")

  for(j in 1:90) {
    pred1 <- nn.pred[j,1]
    predX <- nn.pred[j,3]
    pred2 <- nn.pred[j,2]

    if(pred1 > predX && pred1 > pred2){       preds[j,1] <- "1"
    }else if(predX > pred1 && predX > pred2){ preds[j,1] <- "X"
    }else if(pred2 > pred1 && pred2 > predX){ preds[j,1] <- "2" }
  }

  # ----------------------- Arbol J48 ----------------------------------

  ar.predict <- predict(arbolJ48, newdata = dtest)

  # ---------------- Creacion del dataframe -----------------------------

  A <- data.frame(preds[,1],ar.predict,dtest[,ncol(dtest)])
  names(A) <- c("RedNeur","Arbol","quiniela")

  # ------------- Arbol J48 dispuesto en serie --------------------------

  final.predict <- predict(arbol.Final, newdata = A)
  t <- table(final.predict,A$quiniela)

  Visualizacion(t,nrows)
}
```

- **TFG 2; Paralelo: 1 red neuronal y 1 J48; Serie: 1 red neuronal**

Este diseño es idéntico al anterior con la salvedad de que el clasificador dispuesto en serie es una red neuronal en vez de un árbol de decisión J48.

La implementación también es igual, salvo que el último clasificador pasa a ser una red neuronal, por lo que en la predicción hemos de escoger la clase de mayor probabilidad a la hora de devolver las predicciones.

```
Entrenamiento <- function(dtrain) {

  ...

  # --------- Finaliza el entrenamiento en paralelo ------------

  A <- data.frame(preds[,1],arbolJ48$predictions,dtrain[,21])
  names(A) <- c("RedNeur","Arbol","quiniela")


  # ----------- Red Neuronal (7) dispuesto en serie ----------------------

  nn.final  <<-  nnet(quiniela~.,  data=A,  size=7,  maxit=500,  decay=1,
trace=FALSE)
  final.predict <- predict(nn.final,A,type="raw")

  for(j in 1:nrows) {
    pred1 <- final.predict[j,1]
    predX <- final.predict[j,3]
    pred2 <- final.predict[j,2]

    if(pred1 > predX && pred1 > pred2){       preds.final[j,1] <- "1"
    }else if(predX > pred1 && predX > pred2){ preds.final[j,1] <- "X"
    }else if(pred2 > pred1 && pred2 > predX){ preds.final[j,1] <- "2" }
  }

  t <- table(preds.final,A$quiniela)
  Visualizacion(t,nrows)
}
```

```
Prediccion <- function(dtest) {

  ...

  # ---------------- Creacion del dataframe --------------------

  A <- data.frame(preds[,1],ar.predict,dtest[,ncol(dtest)])
  names(A) <- c("RedNeur","Arbol","quiniela")
```

```
  # ----------- Red Neuronal (7) dispuesto en serie ----------------------

  final.predict <- predict(nn.final,A,type="raw")

  for(j in 1:nrows) {
    pred1 <- final.predict[j,1]
    predX <- final.predict[j,3]
    pred2 <- final.predict[j,2]

    if(pred1 > predX && pred1 > pred2){       preds.final[j,1] <- "1"
    }else if(predX > pred1 && predX > pred2){ preds.final[j,1] <- "X"
    }else if(pred2 > pred1 && pred2 > predX){ preds.final[j,1] <- "2" }
  }

  t <- table(preds.final,A$quiniela)
  Visualizacion(t,nrows)
}
```

- **TFG 3; Paralelo: 4 redes neuronales (de 5 a 8) y 1 J48; Serie: 1 J48**

Diseño un poco más complejo del sistema.

En este caso tenemos 4 redes neuronales en el primer nivel. Estas difieren las unas de las otras en el número de neuronas en la capa oculta (de 5 a 8).

- **TFG 4; Paralelo: 4 redes neuronales (de 5 a 8) y 1 J48; Serie: 1 red neuronal**

Diseño un poco más complejo del sistema y similar al anterior.

Al igual que en el diseño anterior tenemos 4 redes neuronales en el primer nivel, las cuales difieren en el número de neuronas en la capa oculta (de 5 a 8).

En este caso, el clasificador dispuesto en serie es una red neuronal.

- **TFG 5; Paralelo: 2 redes neuronales (5 y 7) y 2 J48; Serie: 1 J48**

En este caso tenemos solo 2 redes neuronales en el primer nivel, acompañadas de 2 árboles de decisión J48. Las redes neuronales se distinguen por el número de neuronas en la capa oculta (una tiene 5 y la otra 7).

Los árboles de decisión difieren en que uno de ellos esta podado con pre-poda al alcanzar mínimo de 20 instancias al formar el nodo, y el otro esta podado también con pre-poda, pero con un parámetro *M* de 50. De esta forma conseguimos el objetivo de la pluralidad de votos.

- **TFG 6; Paralelo: 2 redes neuronales (5 y 7) y 2 J48; Serie: 1 red neuronal**

Al igual que en el diseño anterior tenemos 2 redes neuronales y 2 árboles de decisión J48 en el primer nivel. Las redes neuronales se distinguen por el número de neuronas en la capa oculta (5 y 7), y los árboles de decisión en los parámetros de control.

En este caso, el clasificador dispuesto en serie es una red neuronal.

- **TFG 7; Paralelo: 3 redes neuronales (5, 6 y 7) y 2 J48; Serie: 1 J48**

En este caso tenemos 3 redes neuronales y 2 árboles de decisión J48 en el primer nivel. Las redes neuronales se distinguen por el número de neuronas en la capa oculta (5, 6 y 7), y los árboles de decisión en los parámetros de control (U y M).

- **TFG 8; Paralelo: 3 redes neuronales (5, 6 y 7) y 2 J48; Serie: 1 red neuronal**

Al igual que en el diseño anterior tenemos 3 redes neuronales y 2 árboles de decisión J48 en el primer nivel. Las redes neuronales se distinguen por el número de neuronas en la capa oculta (5, 6 y 7), y los árboles de decisión en los parámetros de control (U y M).

En este caso, el clasificador dispuesto en serie es una red neuronal.

- **TFG 9; Paralelo: 4 redes neuronales (de 5 a 8) y 2 J48; Serie: 1 J48**

En este caso tenemos 4 redes neuronales, como en el diseño TFG 3, y 2 árboles de decisión J48 en el primer nivel. Las redes neuronales se distinguen por el número de neuronas en la capa oculta (de 5 a 8), y los árboles de decisión en los parámetros de control (U y M) de los que depende el sobreajuste del árbol.

- **TFG 10; Paralelo: 4 redes neuronales (de 5 a 8) y 2 J48; Serie: 1 red neuronal**

Al igual que en el diseño anterior tenemos 4 redes neuronales, como en el diseño TFG 3, y 2 árboles de decisión J48 en el primer nivel. Las redes neuronales se distinguen por el número de neuronas en la capa oculta (de 5 a 8), y los árboles de decisión en los parámetros de control (U y M) de los que depende el sobreajuste.

En este caso, el clasificador dispuesto en serie es una red neuronal.

- **TFG 11; Paralelo: 4 redes neuronales (de 5 a 8) y 3 J48; Serie: 1 J48**

En este caso tenemos 4 redes neuronales, como en el diseño TFG 3, y 3 árboles de decisión J48 en el primer nivel. Las redes neuronales se distinguen por el número de neuronas en la capa oculta (de 5 a 8), y los árboles de decisión en los parámetros de control (U y M) de los que depende el sobreajuste del árbol. Estos parámetros son ahora:

En el árbol 1: U = default (post-poda). M = default (1).

En el árbol 2: U = TRUE (pre-poda). M = 20.

En el árbol 3: U = TRUE (pre-poda). M = 50 (más generalizado que el árbol 2).

- **TFG 12; Paralelo: 4 redes neuronales (de 5 a 8) y 3 J48; Serie: 1 red neuronal**

Al igual que en el diseño anterior tenemos 4 redes neuronales, como en el diseño TFG 3, y 3 árboles de decisión J48 en el primer nivel. Las redes neuronales se distinguen por el número de neuronas en la capa oculta (de 5 a 8), y los árboles de decisión en los parámetros de control (U y M) de los que depende el sobreajuste.

En este caso, el clasificador dispuesto en serie es una red neuronal.

- **TFG 13; Paralelo: 4 redes neuronales (5,5,8,8) y 1 J48; Serie: 1 J48**

Implementación con pequeñas modificaciones del sistema TFG 3.

En este caso tenemos 4 redes neuronales y solo un árbol de decisión J48 en el primer nivel, como en el diseño TFG 3. En este caso difieren ambas implementaciones en los parámetros de las redes neuronales, concretamente en número de neuronas en la capa oculta de cada una de ellas. En TFG 3, estas iban de 5 a 8, sin embargo, aplicamos una modificación y hacemos que estas sean 5 en dos de ellas, y 8 en las dos restantes.

Esta modificación la hacemos para observar en qué medida, cambios en el número de neuronas de la capa oculta de cada red neuronal siendo estos significativos, afectan a la predicción del sistema diseñado.

- **TFG 14; Paralelo: 4 redes neuronales (5,5,8,8) y 1 J48 (M = 50); Serie: 1 J48 y TFG 15; Paralelo: 4 redes neuronales (5,5,8,8) y 1 J48; Serie: 1 J48 (M = 5)**

Ambas son implementaciones con pequeñas modificaciones del sistema TFG 13.

En este caso tenemos 4 redes neuronales y solo un árbol de decisión J48 en el primer nivel, como en el diseño TFG 13. Las tres implementaciones (13, 14, 15) difieren en los parámetros de los árboles de decisión.

En TFG 13, el árbol de decisión dispuesto en paralelo era pre-podado con parámetro de control U = 20, mientras que en la implementación TFG 14 se indica que su ajuste de instancias sea de 50 (M = 50).

```
arbolJ48 <<- J48(quiniela ~ ., data = dtrain,
+                 control = Weka_control(U = TRUE, M = 50))
```

En TFG 13 y 14, el árbol de decisión dispuesto en serie era post-podado, mientras que en la implementación TFG 15 se indica que se lleve a cabo pre-poda (U = TRUE) y que su ajuste de instancias por hoja sea de 5 (M = 5).

```
arbol.Final <<- J48(quiniela ~ ., data = dtrain,
+                    control = Weka_control(U = TRUE, M = 5))
```

Estas modificaciones la hacemos para observar en qué medida, cambios en los parámetros de poda de los árboles de decisión, afectan a la predicción del sistema.

- **TFG 20; Paralelo: 1 red neuronal y 1 J48; Votación mayoritaria simple**

Es el diseño más básico con un método de combinación. En todos estos modelos se ha utilizado la estructura en paralelo dentro de la arquitectura híbrida de diseños anteriores. De esta forma podemos comparar los resultados obtenidos según el método de fusión de información, ya que colocando un clasificador en serie se pretende que este fusione los resultados obtenidos de los clasificadores en paralelo.

En este caso se tiene una arquitectura en paralelo con un método de combinación para obtener una respuesta del sistema. En el entrenamiento se toman los datos del conjunto de entrenamiento y se les pasa a una red neuronal y a un árbol de decisión J48 que actúan en paralelo. De estos se obtiene unos resultados y se pasan a un dataframe intermedio, pero al contrario que en los diseños anteriores, este no pasa a un nuevo clasificador, sino que es recorrido y cada fila pasa a ser la entrada del método de combinación, en este caso, la votación mayoritaria simple, obteniendo un resultado conjunto.

```
Entrenamiento <- function(dtrain) {

  library(nnet)
  library(RWeka)

  nrows <- nrow(dtrain)
  preds <- matrix(0,nrows,1)
  preds.final <<- matrix(0,nrows,1)
```

```
  # ---------------------- Red Neuronal (7) -----------------------------

  nn.fit <- nnet(quiniela~., data=dtrain, size=7, maxit=500, decay=1,
trace=FALSE)
  nn.pred <- predict(nn.fit,dtrain,type="raw")

  for(j in 1:nrows) {
    pred1 <- nn.pred[j,1]
    predX <- nn.pred[j,3]
    pred2 <- nn.pred[j,2]

    if(pred1 > predX && pred1 > pred2){       preds[j,1] <- "1"
    }else if(predX > pred1 && predX > pred2){ preds[j,1] <- "X"
    }else if(pred2 > pred1 && pred2 > predX){ preds[j,1] <- "2" }
  }

  # ---------------------- Arbol J48 ----------------------------------

  arbolJ48 <<- J48(quiniela ~ ., data = dtrain, control = Weka_control(U =
TRUE, M = 20))

  # ---------------- Creacion del dataframe ----------------------------

  A <<- data.frame(preds[,1],arbolJ48$predictions)
  names(A) <<- c("RedNeur","Arbol")

  # ---------------------- Votacion -----------------------------------

  for(i in 1:nrows){
    preds.final[i,1] <<- Votacion(A[i,])
  }

  t <- table(preds.final,dtrain[,ncol(dtrain)])
  Visualizacion(t,nrows)
}
```

A la hora de la predicción se le pasan los datos de prueba a los clasificadores individuales ya entrenados, estos devuelven unos resultados, y por votación mayoritaria simple se obtienen el resultado final. Posteriormente se devuelve el resumen de los resultados.

```
Prediccion <- function(dtest) {

  library(nnet)
  library(RWeka)
```

```
  nrows <- nrow(dtest)
  preds <- matrix(0,nrows,4)
  preds.final <- matrix(0,nrows,1)

  # ---------------------- Red Neuronal (7) -----------------------------

  nn.pred <- predict(nn.fit,dtest,type="raw")

  for(j in 1:90) {
    pred1 <- nn.pred[j,1]
    predX <- nn.pred[j,3]
    pred2 <- nn.pred[j,2]

    if(pred1 > predX && pred1 > pred2){       preds[j,1] <- "1"
    }else if(predX > pred1 && predX > pred2){ preds[j,1] <- "X"
    }else if(pred2 > pred1 && pred2 > predX){ preds[j,1] <- "2" }
  }

  # ---------------------- Arbol J48 ----------------------------------

  ar.predict <- predict(arbolJ48, newdata = dtest)

  # ---------------- Creacion del dataframe -----------------------------

  A <- data.frame(preds[,1],ar.predict)
  names(A) <- c("RedNeur","Arbol")

  # ---------------------- Votacion -----------------------------------

  for(i in 1:nrows){
    preds.final[i,1] <- Votacion(A[i,])
  }

  t <- table(preds.final,dtest[,ncol(dtest)])

  Visualizacion(t,nrows)
}
```

- **TFG 21; Paralelo: 4 redes neuronales (de 5 a 8) y 1 J48; Votación MS**

Diseño idéntico al anterior salvo que, en este caso, tenemos 4 redes neuronales en el primer nivel. Estas difieren las unas de las otras en el número de neuronas en la capa oculta (de 5 a 8).

Con ello pretendemos obtener un mayor número de votos para una misma instancia y así tener más “puntos de vista” a la hora de apoyar un resultado.

- **TFG 22; Paralelo: 2 redes neuronales (5 y 7) y 2 J48; Votación MS**

En este caso tenemos solo 2 redes neuronales en el primer nivel, acompañadas de 2 árboles de decisión J48. Las redes neuronales se distinguen por el número de neuronas en la capa oculta (una tiene 5 y la otra 7).

Los árboles de decisión difieren en que uno de ellos esta podado con pre-poda al alcanzar mínimo de 20 instancias al formar el nodo, y el otro esta podado también con pre-poda, pero con un parámetro *M* de 50. De esta forma conseguimos pluralidad de votos.

- **TFG 23; Paralelo: 4 redes neuronales (de 5 a 8) y 2 J48; Votación MS**

En este caso tenemos 4 redes neuronales y 2 árboles de decisión J48 en el primer nivel. Las redes neuronales se distinguen por el número de neuronas en la capa oculta (de 5 a 8), y los árboles de decisión en los parámetros de control (U y M) de los que depende el sobreajuste del árbol.

- **TFG 24; Paralelo: 4 redes neuronales (de 5 a 8) y 3 J48; Votación MS**

En este caso tenemos 4 redes neuronales y 3 árboles de decisión J48 en el primer nivel. Las redes neuronales se distinguen por el número de neuronas en la capa oculta (de 5 a 8), y los árboles de decisión en los parámetros de control (U y M) de los que depende el sobreajuste del árbol.

- **TFG 33; Paralelo: 4 ANN y 2 J48; Votación mayoritaria por pesos**

En este diseño también se tiene una arquitectura en paralelo con un método de combinación para obtener una respuesta del sistema. La diferencia recae en el método de combinación, que pasa a ser una votación mayoritaria por pesos. Los pesos utilizados para cada clasificador han sido obtenidos mediante un estudio realizado sobre el acierto reflejado en los resultados de cada clasificador para cada una de las clases.

Clase	**ANN 5**	**ANN 7**	**J48**	**J48 20**	**J48 50**
1	0.7	0.5	0.5	0.5	0.5
2	0.4	0.3	0.6	0.7	0.5
X	0.9	0.7	0.4	0.6	0.8

Tabla 2. Pesos para cada clasificador individual

De esta forma hemos obtenido la tabla 2, que representa la matriz de pesos utilizada para la votación, donde el número al lado de las siglas ANN, representa el número

de neuronas en la capa oculta de la red neuronal; y al lado de J48, representa el valor del parámetro de control U (si no tiene, representa post-poda).

Podemos ver en esta matriz de pesos como tenemos clasificadores con un peso mayor para una clase que para otras. Esto no significa que este clasificador haya logrado clasificar bien muchas entradas de esta clase, sino que el porcentaje de acierto de entradas que ha clasificado en ella es alto, es decir, de las que ha clasificado en esa clase, un gran porcentaje de ellas pertenecían realmente a esa clase. Se tiene entonces que el clasificador es fiable a la hora de dar resultados de esa clase, por ello su peso es mayor.

Este es el caso de la red neuronal con 5 neuronas en la capa oculta. El número de instancias clasificadas en la clase *X* era de 275, clasificando este solo 18 como *X*. Pero de esos 18 que clasificó como *X*, acertó en 14, por lo que es muy fiable. A la hora de la votación, si este clasificador dice que la instancia se clasifica como *X* es más probable que acierte, por lo que su peso debe ser mayor para apoyar esta idea.

Así, para cada entrada del método de combinación comprobamos el peso de los votos según el clasificador que lo haya devuelto, y la clase cuya suma de votos sea mayor, es la clase que se devuelve como resultado del conjunto.

- **TFG 34; Paralelo: 4 ANN y 3 J48; Votación mayoritaria por pesos**

En este caso también tenemos una votación mayoritaria por pesos, pero ahora disponemos de 3 árboles de decisión J48. Uno de ellos creado mediante post-poda, otro creado por pre-poda con parámetro de control igual a 20, y otro con pre-poda y parámetro de control igual a 50.

- **TFG 35; Paralelo: 4 ANN y 3 J48; Votación mayoritaria por pesos ajustables**

En este diseño se trabaja con una votación mayoritaria por pesos como método de combinación, pero se cambian pesos fijos por pesos ajustables, es decir, los pesos van cambiando según se construyen los resultados con ellos. De esta forma, lo que pretendemos es que el propio sistema ajuste los pesos según su porcentaje de éxito.

El modelo empieza con todos los pesos iguales, y con la primera entrada, al comenzar el entrenamiento, el modelo realiza una combinación de resultados equivalente al voto mayoritario simple. A partir de esta primera entrada los pesos empiezan a cambiar. Para un porcentaje de los datos de entrenamiento, al inicio de estos, los resultados obtenidos se suponen de menor calidad, ya que los pesos no son fiables, sin embargo, en el trascurso del entrenamiento, los pesos van ajustándose a los resultados obtenidos, por lo que los resultados devueltos tras la combinación son cada vez de mayor calidad.

Con este propósito hemos creado un algoritmo de votación por pesos tal como sigue:

ENTRADA resultados de clasificadores individuales para instancia (*row*)

Inicializamos *ncolm*s como el número de clasificadores utilizados para la creación de *row,* siendo *row[,ncolms+1]* la clase real (aprend. supervisado)

Inicializamos los votos a 0, votos1 = 0, votos2 = 0, votosX = 0

PARA i = 1

SI row[,i] == "1" el voto es 1
ENTONCES votos1 = votos1 + matrizPesos[1,i]

SI row[,i] == "2"
ENTONCES votos2 = votos2 + matrizPesos[2,i]

SI row[,i] == "X"
ENTONCES votosX = votosX + matrizPesos[3,i]

SI row[,i] == row[,ncolms+1] es correcto el resultado
ENTONCES

SI es clase 1
ENTONCES matrizPesos[1,i] = matrizPesos[1,i] + 0.01

SI es clase 2
ENTONCES matrizPesos[2,i] = matrizPesos[2,i] + 0.02

SI es clase X
ENTONCES matrizPesos[3,i] = matrizPesos[3,i] + 0.03

SI NO

SI el resultado del clasificador es clase 1
ENTONCES matrizPesos[1,i] = matrizPesos[1,i] - 0.01

SI el resultado del clasificador es clase 2
ENTONCES matrizPesos[2,i] = matrizPesos[2,i] - 0.01

SI el resultado del clasificador es clase X
ENTONCES matrizPesos[3,i] = matrizPesos[3,i] - 0.01

HASTA ncolms

SALIDA clase con mayoría de votos, *max(votos1, votos2, votosX)*

Vemos que los pesos van cambiando atendiendo a dos razones, si el resultado devuelto por el clasificador ha sido correcto, y cuál es la clase en la que se ha clasificado. Se ha decidido hacer de esta forma para favorecer a aquellos clasificadores individuales que sean capaces de clasificar correctamente y de forma constante las clases "*2*" y "*X*".

Como ya se ha visto, el número de instancias clasificadas como clase "*1*" es superior al resto de clases, por lo que sus pesos aumentarán rápidamente. Para contrarrestar esto, hacemos que acertar en las demás clases ofrezca un mayor beneficio e instancias posteriores puedan optar más fácilmente a ser clasificadas en el resultado final con una clase que no sea "*1*".

4.3. Otros diseños

Como ya se explicó en el apartado de los multiclasificadores, podemos distinguir unos de otros no solo por número de clasificadores individuales integrados o por el tipo de los mismos, sino también por la naturaleza o el tamaño de los conjuntos de datos de entrenamiento usados en cada clasificador.

Como parte de las pruebas de diseño, han sido diseñados tres multiclasificadores los cuales se diferencian de los anteriores en que no varían en la cantidad o la naturaleza de los clasificadores individuales, sino en los datos que recibe cada uno.

- En el primero, se hacen particiones aleatorias en las instancias del conjunto de datos, y cada partición se le pasa como entrada a un clasificador individual.
- En el segundo, se hacen particiones en los atributos. Se crean conjuntos de datos cada uno con atributos distintos, intentando hacer que los clasificadores sean capaces de aprender a clasificar teniendo en cuenta atributos diferentes.
- En el tercero, llamado **TFG 40**, se crean réplicas del conjunto de datos, haciendo que cada uno de ellos tenga una lógica bievaluada. De esta forma se tienen tres conjuntos de datos, uno con clases "*1*" y "*No 1*", otro con clases "*2*" y "*No 2*" y un tercero con clases "*X*" y "*No X*", que son tratados por diferentes clasificadores. Con esto tratamos que cada clasificador se centre en identificar las diferencias entre pertenecer o no a una clase.

  Los resultados obtenidos por los clasificadores individuales pasan a ser evaluados por un método de votación mayoritaria especial, en la cual, si el resultado es "*1*", "*2*" o "*X*" se cuenta como un voto, pero si el resultado en "*No 1*", "*No 2*" o "*No X*", se cuenta como un voto a las dos clases contrarias. Es decir, si el clasificador dice que es "*No 1*", contamos un voto para la clase "*2*" y para la clase "*X*", y así con las otras negaciones. De los tres diseños anteriores, es con el que mejor porcentaje de acierto se obtiene.

5. Análisis de resultados

Para la evaluación de los modelos comentados en el apartado anterior, se han utilizado un conjunto de datos de entrenamiento (*training set*) y un conjunto de datos de prueba o test (*test set*). Estos han sido creados a partir del conjunto de datos original, que como ya ha sido comentado, contiene como instancias los atributos de los partidos de la liga profesional española de fútbol (Liga BBVA) jugados entre la temporada 2004/2005 y la temporada 20014/2015, esta última corresponde a la temporada que finalizó en mayo de este año.

Como método de evaluación ha sido elegida la validación simple [1]. En este método se reserva un porcentaje de los datos como conjunto de prueba, sin que estos sean usados a la hora de construir el modelo. La división de los datos en los conjuntos debe ser aleatoria para una estimación correcta, y el porcentaje de conjunto de prueba suele estar entre el 5 y el 50%. En nuestro caso, se ha utilizado un porcentaje del 25% para el conjunto de prueba, lo que deja un 75% para el conjunto de datos de entrenamiento. De esta forma tenemos 4180 instancias del conjunto de datos total, 3160 en el conjunto de entrenamiento y 1020 en el de prueba.

Según la tarea de minería de datos que se lleve a cabo, existen diferentes medidas de evaluación de los modelos creados. En el contexto de la clasificación lo normal es evaluar la calidad de los modelos con respecto a su precisión predictiva [1], de forma que se calcula como el número de instancias totales en el conjunto de prueba clasificadas correctamente, dividido por el número de instancias totales en el conjunto de prueba. Para la interpretación del modelo, se tiene en cuenta la clasificación en el conjunto de prueba, y no en el conjunto de entrenamiento, debido a que es trivial tener una precisión predictiva de 100% en el conjunto de entrenamiento. Bastaría que el modelo creara una regla para cada instancia para conseguir el 100% con él.

Como medida de evaluación de los modelos creados se va a usar esta precisión predictiva. Sin embargo, no solo se va a utilizar el conjunto de prueba para la interpretación, sino que también se va a utilizar el conjunto de entrenamiento para poder observar cuanto se ajustan los modelos a los datos de entrenamiento, y ver si existe algún tipo de relación entre las precisiones conseguidas con cada conjunto.

En la Tabla 3 se pueden observar las medidas de evaluación obtenidas para cada uno de las implementaciones de los diseños descritos en el apartado anterior (para mayor comodidad, si se pincha con el ratón en el nombre de un diseño, le lleva directamente a la descripción del mismo en el apartado anterior). Estas medidas representan el máximo de las medidas obtenidas tras 10 entrenamientos distintos de los modelos. Esto es debido a que los árboles de decisión sí son deterministas, es decir, para un mismo conjunto de entrenamiento se obtiene siempre el mismo resultado; sin embargo, las redes neuronales del paquete *nnet* no lo son, por lo que cada vez que las entrenamos, el resultado y las medidas obtenidas cambian.

5.1. Precisión Predictiva

Diseño	Nº aciertos entrenamiento	% aciertos entrenamiento	Nº aciertos prueba	% aciertos prueba
J48	1819	57.56	516	50.58
ANN	1983	62.75	480	47.05
AdaBoost	1690	53.48	524	51.37
Bagging	1799	56.93	520	50.98
TFG 1	2016	63.79	458	44.91
TFG 2	1991	63.01	512	50.19
TFG 3	2137	67.62	470	46.07
TFG 4	2111	66.80	472	46.27
TFG 5	2031	64.27	512	50.19
TFG 6	2015	63.76	497	48.72
TFG 7	2049	64.84	502	49.21
TFG 8	2029	64.20	511	50.09
TFG 9	2128	67.35	499	48.92
TFG 10	2094	66.26	501	49.11
TFG 11	2187	69.20	498	48.82
TFG 12	2142	67.78	490	48.03
TFG 13	2072	65.56	491	48.13
TFG 14	2124	67.21	484	47.38
TFG 15	2173	68.76	483	47.35
TFG 20	1860	58.86	522	51.17
TFG 21	1819	57.56	526	51.56
TFG 22	1874	59.30	540	52.94
TFG 23	1956	61.89	537	52.64
TFG 24	1947	61.61	531	52.05
TFG 33	1979	62.62	544	53.33
TFG 34	1919	60.72	530	51.96
TFG 35	1950	61.70	515	50.49
TFG 40	1932	61.13	521	51.08

Tabla 3. Medidas de evaluación obtenidas

En la Tabla 3 tenemos dos columnas orientadas a los resultados obtenidos con los datos de entrenamiento y dos para los datos de prueba. A su vez, dos de ellas nos muestran el número de aciertos obtenidos, y otras dos que muestran la precisión predictiva del modelo utilizado. Estas últimas aparecen indicadas con el nombre de porcentaje de aciertos *(% aciertos*) y el conjunto de datos utilizado.

Como podemos observar mejor en la Figura 11, todos los diseños presentan una mayor precisión predictiva cuando se trata con los propios datos de entrenamiento que cuando se trata con los datos de prueba. Esta es una conclusión lógica, ya que, como ya ha sido comentado antes, los modelos suelen ajustarse a los datos de entrenamiento cuando se van creando.

También podemos observar en la Figura 11, que el número de modelos que supera el 50% de precisión predictiva con el conjunto de pruebas, es de 15, lo que supone más de la mitad de los modelos. Sin embargo, vemos que ninguno de ellos sobrepasa el 55%, por lo que la calidad predictiva es baja. El modelo de referencia es el AdaBoost.M1, ya que, como vimos anteriormente, ganó el "Premio Gödel" en 2003, y es el multiclasificador con el que mayor porcentaje de acierto hemos obtenido. Partiendo de esto, podemos ver que 6 modelos superan el porcentaje obtenido con AdaBoost.M1, si bien no lo superan en exceso.

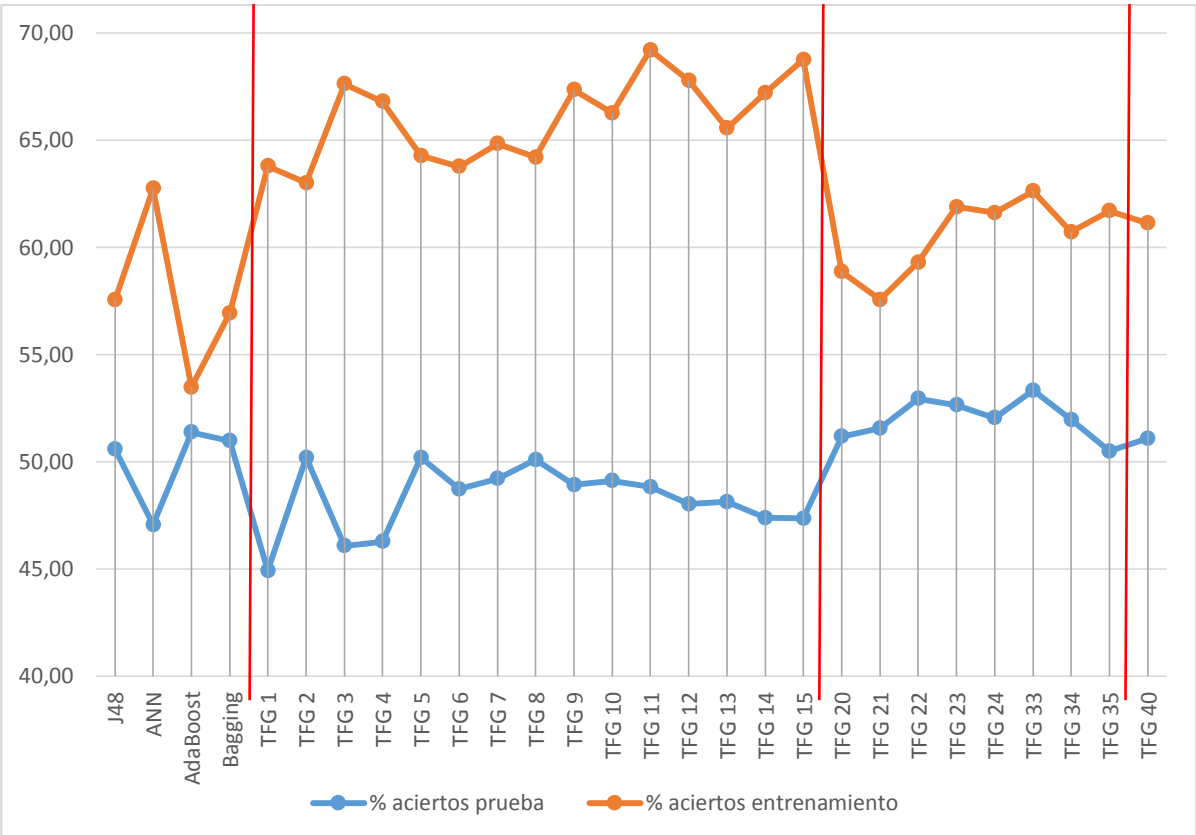

Figura 11. Precisión predictiva con los diferentes conjuntos

Como vemos, hay unas separaciones rojas en las Figuras 11 y 12, que representan la división realizada a la hora de estructurar los diseños. Se puede observar en la Figura 11 como los modelos con una arquitectura híbrida no ofrecen buenos resultados, al contrario que las arquitecturas paralelas con los diferentes métodos de combinación. De los primeros, sólo 3 superan (por no más de 0.2%) el 50% de aciertos con el conjunto de pruebas, y ninguno de ellos supera el obtenido por un clasificador simple como el árbol de decisión J48 o C4.5.

En la Figura 12 se representa la diferencia de precisión predictiva conseguida con el conjunto de datos de entrenamiento y el conjunto de datos de prueba. Podemos observar como los modelos con una arquitectura híbrida (de TFG 1 a TFG 15) presentan la mayor diferencia de acierto, al igual que la red neuronal artificial. Si nos fijamos también en la figura anterior, podemos apreciar que es debido a su poca generalización. Esto quiere decir que el modelo se ajusta demasiado a los datos de entrenamiento, y a la hora de clasificar datos nuevos no es capaz de predecirlos bien.

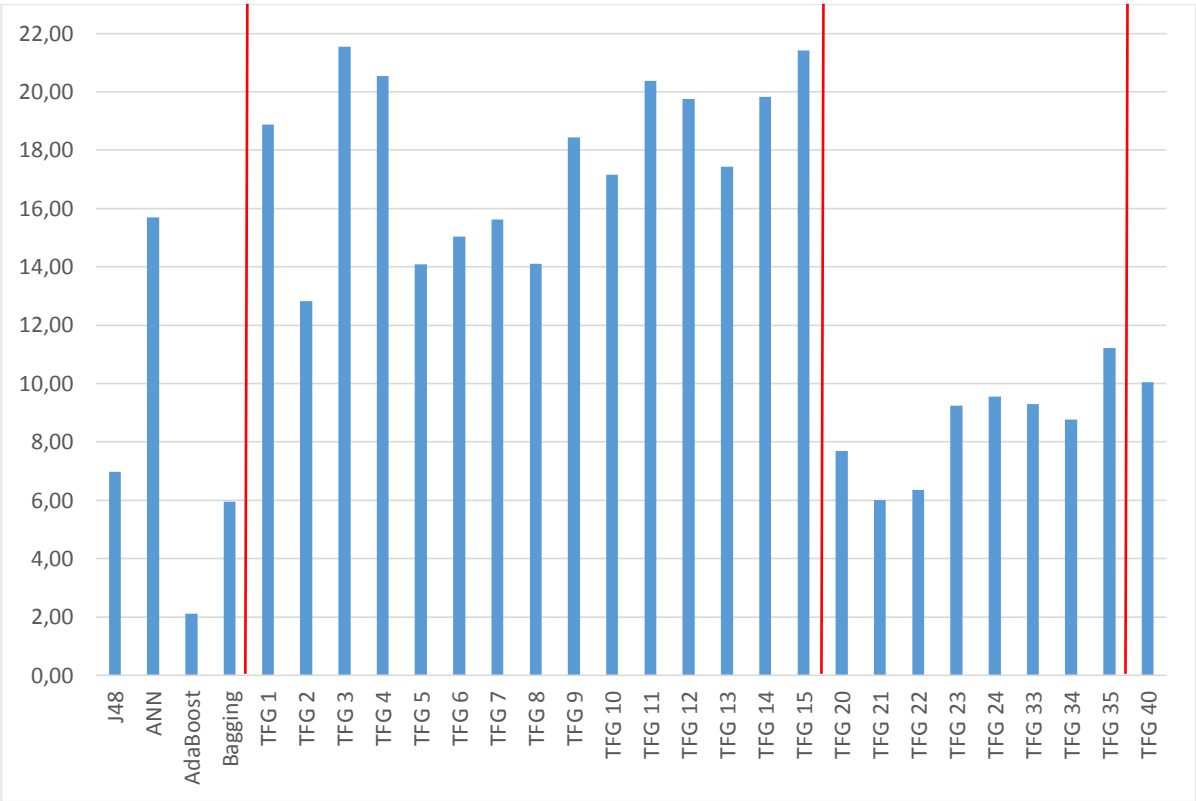

Figura 12. Diferencia en el porcentaje de acierto

En ningún caso significa que el de menor diferencia en el porcentaje de acierto sea el de mayor precisión, ya que, como podemos ver en la Figura 12, el modelo AdaBoost.M1 es el de menor diferencia, sin embargo, el TFG 33 tiene una mayor precisión predictiva. La diferencia nos indica el grado de generalización del modelo.

5.2. Matriz de confusión

En este caso vamos a usar como medida de evaluación la matriz de confusión obtenida de los modelos. La matriz de confusión es una de las herramientas de visualización más empleada en aprendizaje supervisado [2]. En ella se presentan los valores predichos por el modelo o método, en contraposición con los valores reales. De este modo tenemos que cada columna representa el número de instancias que han sido clasificadas por el modelo en una clase, y cada fila representa esas instancias en la clase real a la que pertenecen en el conjunto de datos.

Uno de los objetivos a la hora de utilizar una matriz de confusión como medida de evaluación, es facilitar la interpretación visual de los resultados devueltos por el sistema y poder observar si este confunde ciertas clases. En el proceso KDD [1], la interpretación de los resultados puede ser más importante que la propia evaluación de los modelos. Esto se ve claramente en el siguiente ejemplo:

"Se tiene un sistema que trabaja en la sala de urgencias de un centro médico, de forma que, ante la entrada de un nuevo paciente, tiene que decidir si se le da el alta u hospitalizarlo. El sistema tras el entrenamiento, presenta una precisión predictiva del 95%. Sin embargo, clasifica todas las entradas dándole el alta al paciente. Ha obtenido el 95% de precisión debido a que el 95% de datos de entrada son altas."

Esa medida de evaluación del 95% no significa que nuestro sistema sea bueno, ya que pone en riesgo mortal al 5% de los pacientes al no hospitalizarlos. Por ello, los resultados han de ser interpretados para comprobar la calidad de los modelos.

Para evaluar e interpretar los modelos, presentamos en la Figura 13 las matrices de confusión de 6 modelos característicos de nuestros diseños. No creemos que haga falta presentar la matriz obtenida en cada diseño, ya que dentro de las divisiones hechas anteriormente no hay cambios representativos. Estas matrices de confusión han sido obtenidas al mismo tiempo que las medidas de evaluación de la Tabla 3.

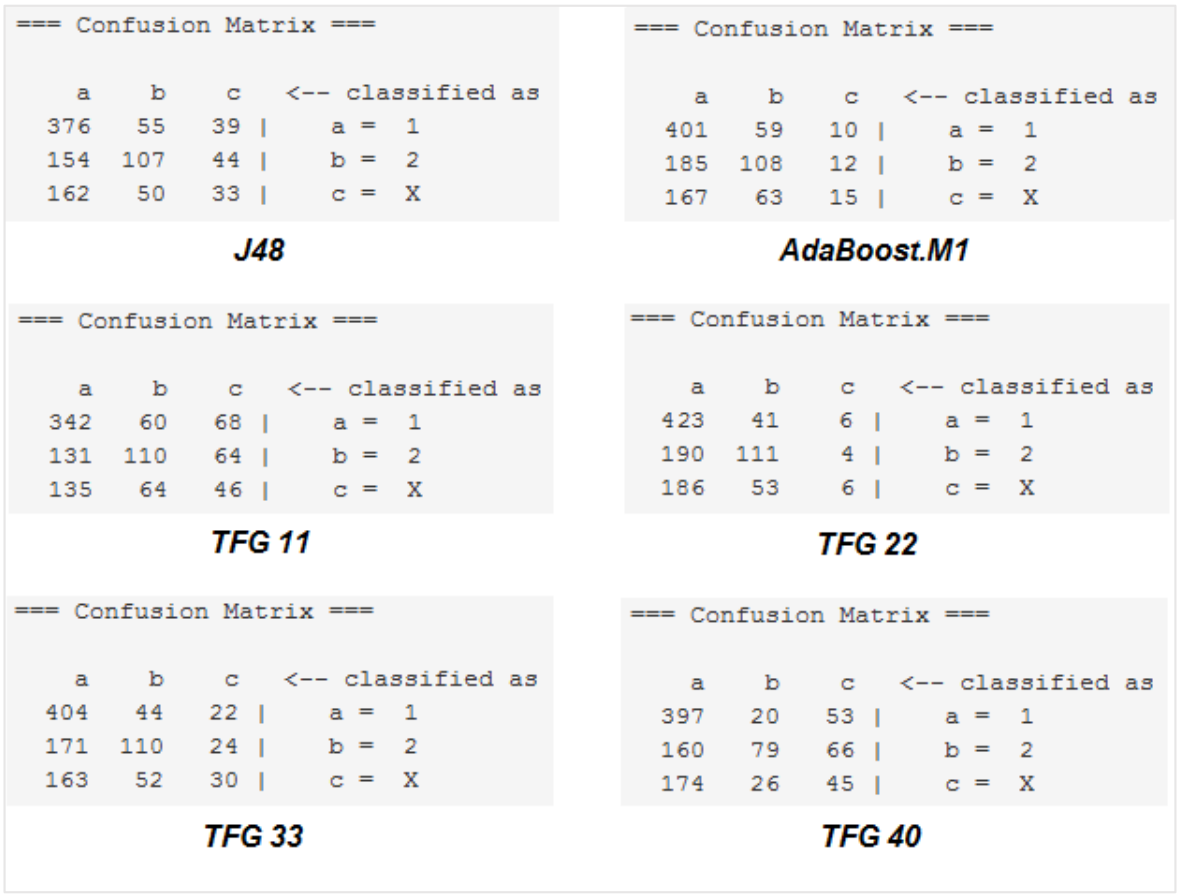

Figura 13. Matriz de confusión de diferentes modelos

Como podemos apreciar en la Figura 13, la mayor parte de las matrices de confusión son muy similares. Vemos que todas ellas obtienen un buen resultado clasificando las instancias de la clase *1*. El que menor error comete en este sentido es el modelo TFG 22 con sólo 47 instancias de esta clase mal clasificadas. Sin embargo, este modelo no es de calidad a la hora de predecir la clase *X*, al igual que pasa con el algoritmo AdaBoost.M1. Ambos obtienen un pequeño número de *X's*, por lo que les ocurre como en el ejemplo del hospital, tienen poco en cuenta esta clase.

La clave de una buena predicción, y un buen modelo, es la clasificación equitativa, obtener buenos resultados en todas las clases. No deben centrarse en una como TTFG 22, pero tampoco cometer demasiados errores intentando clasificar instancias en una clase en concreto, como ocurre con los modelos TFG 11, TFG 40 y el árbol de decisión J48, en los cuales se puede observar cómo estos clasifican muchas instancias en la clase *X*, pero lo hacen incorrectamente. Todos ellos cometen pocos errores en las instancias que predicen como *2*, y cómo podemos observar, el número de instancias correctas de esta clase es similar en todos (110 aprox.).

El modelo que podemos interpretar como el de mejor calidad es el TFG 33. Como podemos observar en la Figura 13, es el que comete un número de errores inferior en las instancias que clasifica, es decir, si clasifica una entrada en una clase, es menos probable que haya cometido un error. Podemos ver en la Figura 14 la matriz de confusión obtenida, pero de forma más visual, donde cada barra representa el número de instancias según la clasificación obtenida de cada clase real.

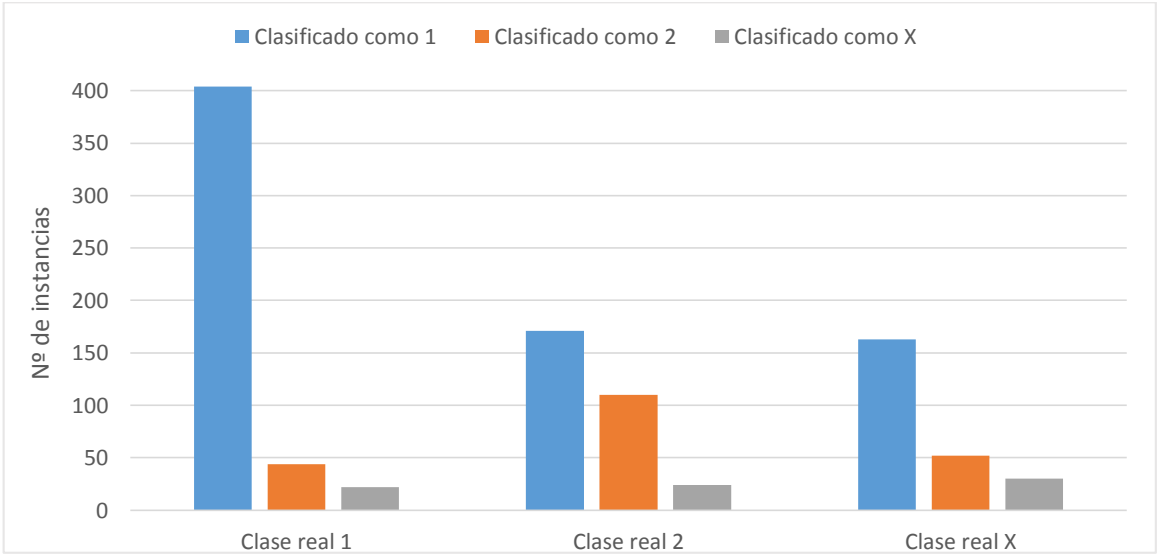

Figura 14. Matriz de confusión de TFG 33 de forma gráfica

Vemos entonces que el modelo TFG 33 no solo es el que mejor precisión predictiva presenta, sino también la mayor calidad, aunque queda lejos de ser un sistema fiable, con una precisión del 53,33%. Sin embargo, que algoritmos fiables y muy utilizados, como son los métodos AdaBoost.M1 y Bagging den medidas de evaluación similares nos hace pensar que existe un límite en el potencial predictivo del propio problema.

6. Conclusiones

A la vista de los resultados obtenidos en el apartado anterior, así como de las interpretaciones que se han hecho, se pueden extraer las siguientes concusiones:

- Se ha conseguido crear varios modelos que, sin sustentarse en métodos teóricos como lo están métodos multiclasificadores basados en boosting [5], como son las técnicas Bagging y AdaBoost.M1, son capaces de obtener resultados similares e incluso superiores que estos.

- Lo anterior puede ser debido en parte a que son modelos o técnicas que han sido desarrolladas para este problema en concreto, con lo cual, no son modelos generalizados por lo que seguramente si se aplicaran a otros conjuntos de datos, no se obtendrían resultados similares.

- No se ha conseguido un ratio de acierto superior al 55% con ninguna técnica de aprendizaje computacional. Este suceso puede encontrar su fuente en dos elementos: los datos utilizados, o el problema en sí.

  Como recoge el capítulo 2, *Descripción del Problema*, hay sucesos que no han podido ser recogidos en los datos utilizados, en concreto, todos aquellos referentes a individualidades, lesiones, ausencias, número de tarjetas. La calidad de las muestras puede ser de vital importancia a la hora de obtener mejores resultados, sin embargo, puede que en este caso, la propia aleatoriedad del problema presente un límite en el acierto. Los resultados en deportes, y más si son de equipo, dependen a veces más de la moral o las sensaciones de los deportistas en la competición, que de la estadística.

- Tal y como se dijo, la opinión de diversos expertos es que una persona con conocimientos normales de la primera división española de fútbol, obtendría una precisión predictiva cercana al 40%, y una persona considerada experta, una precisión sobre el 50%. Esto significa que los modelos utilizados en aprendizaje automático y los diseñados en este trabajo obtienen una reducida mejora de lo que un humano es capaz de obtener.

- Aun así, los resultados obtenidos suponen una mejora respecto a los métodos estadísticos usados actualmente por las casas de apuestas, con los cuales, según un estudio [7], se obtiene un 40% de precisión predictiva.

- Como posible trabajo futuro, se podría demostrar empíricamente que las técnicas diseñadas en el trabajo sufren un aumento en el porcentaje de acierto en las jornadas que se juegan a mitad de temporada. En un principio, este razonamiento parece lógico. Este suceso también ocurre con la gente de la calle y con los expertos, cuando más se avanza en la competición, más datos hay disponibles para comparar.

Se puede ver más claro con el siguiente ejemplo: "*Se tiene un aficionado de la Real Sociedad de Fútbol. Este aficionado, al empezar la liga, siempre apostaría a que su equipo va a ganar si se enfrenta a un equipo de un nivel similar. Van avanzando las jornadas y su equipo va perdiendo partidos, aunque el sigue confiado en que el próximo van a ganar. Cuando la competición va por la jornada 16 (de 38 jornadas) y su equipo ha perdido 10 de esos encuentros consecutivamente, lo más seguro es qué, si este aficionado ha de apostar su dinero por quién va a ganar el próximo partido de su equipo, apueste por el contrincante*."

Porque aunque el aficionado siga sintiendo los colores de su equipo, si se tiene que jugar su dinero, tiene suficientes datos como para replantearse la situación y poder dar un resultado correcto. Algo similar pasa con nuestras técnicas de aprendizaje. Al comenzar la temporada no se tiene ninguna información de los equipos respecto a puntos o goles, sólo respecto a sus presupuestos y su rendimiento en la temporada anterior, con la incógnita de cómo van reaccionarán los equipos recién ascendidos a la competición. Mientras más datos para comparar reciben, mejor es su aprendizaje.

- Otra idea para posibles trabajos futuros debe ser la de plantearse la búsqueda de un algoritmo que produzca de forma estable porcentajes de aciertos tan elevados como sea posible, añadiendo otros clasificadores junto a los ya estudiados, como pueden ser las Máquinas de Soporte Vectorial.

- Por último, podríamos plantearnos la idea de mejorar el conjunto de datos utilizados. Se podría abordar desde distintos frentes: añadir nuevos atributos, algunos sobre el rendimiento de los jugadores; incrementar el número de temporadas que se tienen en cuenta; o ampliar el dominio de los datos a partidos de otras ligas de fútbol profesional, como la segunda división española o la Premier League inglesa.

Referencias

[1] José Hernández Orallo, M.José Ramírez Quintana, Cèsar Ferri Ramírez (2004), "*Introducción a la Minería de datos*". Editorial Pearson.

[2] Jiawei Han and Micheline Kamber (2006), "*Data Mining: Concepts and Techniques*". Morgan Kaufmann.

[3] Saddys Segrera Francia, María N. Moreno García (2006). "*Multiclasificadores: Métodos y Arquitecturas*", Technical Report DPTOIA-IT-2006-001

[4] Sašo Džeroski, Bernard Ženko (2002), "*Is Combining Classifiers Better than Selecting the Best One?"*. Department of Intelligent Systems, Jožef Stefan Institute.

[5] Harris Drucker, Corinna Cortes, L.D. Jackel, Yann LeCun, Vladimir Vapnik (1994), "*Boosting and other ensembles methods*". Neural Computation, vol. 6.

[6] Leo Breiman (1996), "*Bagging predictors*". Machine Learning, Kluwer Academic Publishers, Boston, Manufactured in The Netherlands, vol. 24.

[7] Esteban Alfaro, Matías Gamez, Noelía García (2013), "*adabag: An R Package for Classication with Boosting and Bagging*". Journal of Statistical Software. Volume 54.

[8] Yoav Freund, Robert E. Schapire (1996), "*Experiments with a new boosting algorithm*". Proceedings 13th International Conference on Machine Learning.

[9] Leo Breiman (2001), "*Random Forests*". Machine Learning, vol. 45.

[10] M. Baxter, R. Stevenson (1988), "*Discriminating between the Poisson and negative binomial distributions: an application to goal scoring in association football*". J. Applied Statistics 15, 347-354.

[11] I. McHale, P. Scarf (2006), "*Forecasting international soccer match results using bivariate discrete distributions*". Salford Business School Working Paper N° 322/06

[12] L.M. Hvattum, H. Arntzen (2010), "*Using ELO rating for match result prediction in association football*". International Journal of Forecasting Vol. 26 pp. 460-470

[13] Ángel Calleja (2014), "*El dinero que los madrileños se juegan en apuestas aumenta un 827% en cuatro años*". Periódico 20 Minutos.

[14] Kurt Badenhausen (2015), "*The World's 50 Most Valuable Sports Teams 2015*". Revista Forbes.

[15] J. R. Quinlan, Morgan Kaufmann (1993), "*C4.5: programs for machine learning*".

[16] Brian D. Ripley (1996), "*Pattern Recognition and Neural Networks*". Cambridge.

[17] Andy Liaw (2015). "*Breiman and Cutler's random forests for classification*".

[18] Kurt Hornik, Achim Zeileis, Torsten Hothorn, Christian Buchta (2015), "*R/Weka interface. R package version 0.4-24*".

[19] Brian Ripley, William Venables (2015), "*nnet. R package version 7.3-11*".

Anexo

1. Instalación de R

Para todas las implementaciones del trabajo, se utiliza el lenguaje de programación para análisis estadístico y gráfico, R. En primer lugar hemos de instalar R en nuestro equipo. La versión de R usada es la 3.1.3 para Windows, ya que es la versión estable para la que fueron diseñados distintos paquetes usados, y el equipo que ha sido utilizado para la realización del trabajo tiene sistema operativo Windows. Para ello basta con instalar el ejecutable que aparece en la web:

https://cran.r-project.org/bin/windows/base/old/3.1.3/

Con ello, nos descargaremos el archivo al ordenador para posteriormente ejecutarlo. Una vez instalado este, vamos a instalar una interfaz gráfica llamada RStudio. RStudio es un entorno de desarrollo integrado (IDE) para R que funciona con la versión estándar de R disponible en CRAN. Al igual que R, es software libre. El objetivo de sus creadores era desarrollar una herramienta potente que soporte los procedimientos y técnicas para realizar análisis de alta calidad, y al mismo tiempo, que fuera sencillo e intuitivo para proporcionar un entorno amigable, tanto para los ya experimentados como para los nuevos usuarios de R. Para poder instalarlo hemos de descargarnos el ejecutable del sitio web:

https://www.rstudio.com/products/rstudio/download/

Una vez concluida la instalación ya podemos empezar a utilizar este IDE, cuya apariencia se muestra en la imagen (Fig. 15).

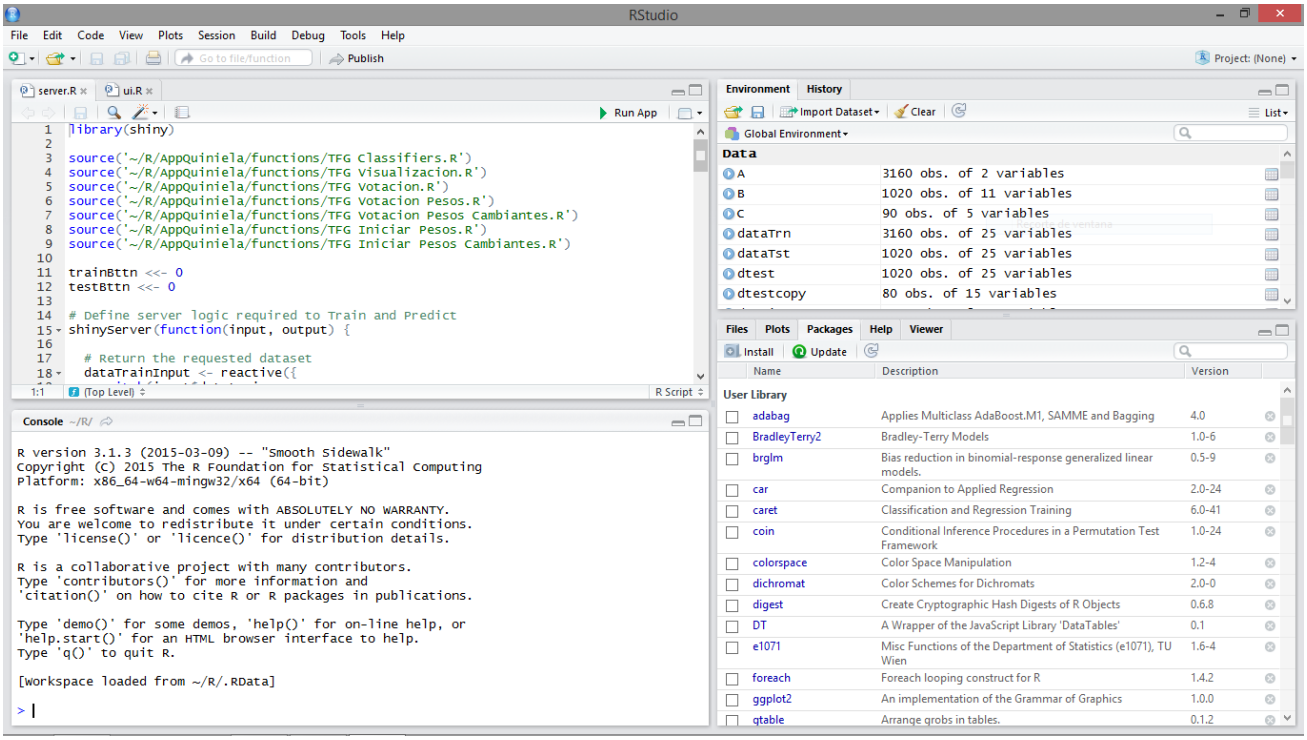

Figura 15. Interfaz de desarrollo de RStudio

2. Paquetes R

Cuando se quiere instalar un paquete R, RStudio trae un instalador propio, el cual ofrece dos opciones para ello. La primera es la más sencilla, consiste en escribir el nombre del paquete en este instalador. Para la segunda hemos de descargarnos el paquete R comprimido (.tar.gz) e indicárselo al instalador. Esta última es la más eficaz, ya que no necesita buscar el paquete en los distintos espejos de los servidores de R. Los paquetes R utilizados para la realización del trabajo son:

- **RWekajars:** este paquete R contiene los métodos y técnicas presentes en el famoso software de minería de datos Weka. Es una colección de algoritmos de aprendizaje automático que contiene herramientas para pre-procesamiento de datos, clasificación, regresión, clustering, reglas de asociación, y de visualización. Podemos descargar este paquete desde el sitio web:

  https://cran.r-project.org/web/packages/RWekajars/index.html

- **RWeka:** es un paquete R que contiene el código de la interfaz para utilizar los métodos y técnicas que se encuentran en el paquete RWekajars. Se puede descargar desde:

  https://cran.r-project.org/web/packages/RWeka/index.html

- **nnet:** es un software para desarrollar redes neuronales feed-forward con una sola capa oculta, y para los modelos log-lineales multinomiales. Podemos descargar este paquete desde el sitio web:

  https://cran.r-project.org/web/packages/nnet/index.html

- **shiny:** es un paquete R para el desarrollo de aplicaciones web interactivas (apps) directamente en el lenguaje de programación R. Para el desarrollo de una aplicación hacen falta dos componentes:
  - **ui.R:** es el programa R dedicado a la interfaz de usuario, el cual controla el diseño y el aspecto de la aplicación.
  - **server.R:** es el programa que hace las funciones de servidor, contiene las instrucciones que el equipo necesita para construir la aplicación.

  Para que se puedan ejecutar aplicaciones shiny, necesitamos instalar dos paquetes más, el paquete **Rcpp** y el paquete **httpuv**. Finalmente, para la ejecutar la aplicación, ambos programas R deben estar juntos en una carpeta, llamada "*NombreApp*", la cual debe estar en el *workspace* de R, y bastará con introducir el comando `shiny::runApp('NombreApp')` en la consola de R.

  Los sitios web desde los que podemos descargar estos paquetes son:

  https://cran.r-project.org/web/packages/shiny/index.html

  https://cran.r-project.org/web/packages/Rcpp/index.html

  https://cran.r-project.org/web/packages/httpuv/index.html

3. Aplicación: Presentación de los Datos

Esta aplicación que hemos desarrollado se encuentra bajo el nombre de "*AppData*". En ella se pueden observar los conjuntos de datos que utilizados a lo largo del trabajo. Para ello, se muestran en la aplicación en diferentes pestañas, una para cada conjunto de datos. Se pueden observar en tablas, donde cada fila representa una instancia y cada columna representa un atributo de los datos. En un principio estas instancias se encuentran ordenadas por orden cronológico.

La aplicación nos permite (Fig. 16):

1. Elegir los atributos que queremos observar de los datos.
2. Elegir el número de entradas que queremos que aparezcan en una página.
3. Navegar entre las diferentes páginas que nos muestran.
4. Filtrar por búsqueda en los datos.
5. Ordenar las instancias por alguno de sus atributos.

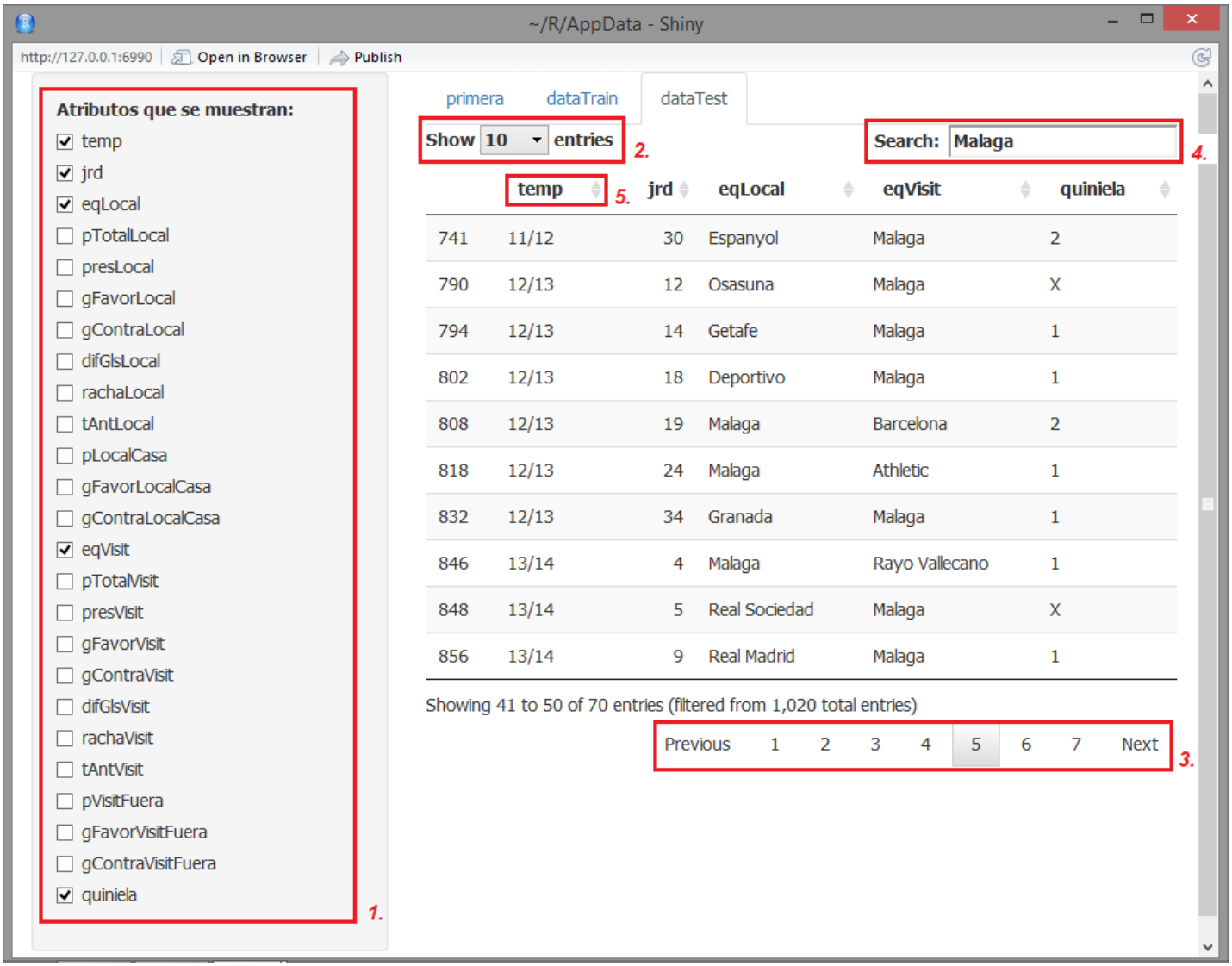

Figura 16. Ejemplo de ejecución de la aplicación AppData

4. Aplicación: Entrenamiento y Predicción

Esta aplicación se encuentra bajo el nombre de "*AppQuiniela*". Es la aplicación que hemos desarrollado para poder hacer las pruebas con las implementaciones realizadas a lo largo de todo el trabajo. Esta aplicación tiene dos pestañas, *Pruebas* y *Quinielas*. Antes de acceder a la pestaña *Quinielas* se necesita entrenar un modelo.

En la primera pestaña nos encontramos con todo lo necesario para entrenar y predecir las diferentes técnicas de aprendizaje (Fig. 17). La aplicación nos permite:

1. Elegir el conjunto de datos utilizado para el entrenamiento.
2. Elegir el conjunto de datos utilizado para las pruebas.
3. Decidir la técnica de aprendizaje automático que queremos probar.
4. Botón de acción para iniciar el entrenamiento.
5. Botón de acción para iniciar las pruebas.
6. Visualizar los resultados obtenidos del entrenamiento.
7. Visualizar los resultados obtenidos de las pruebas.

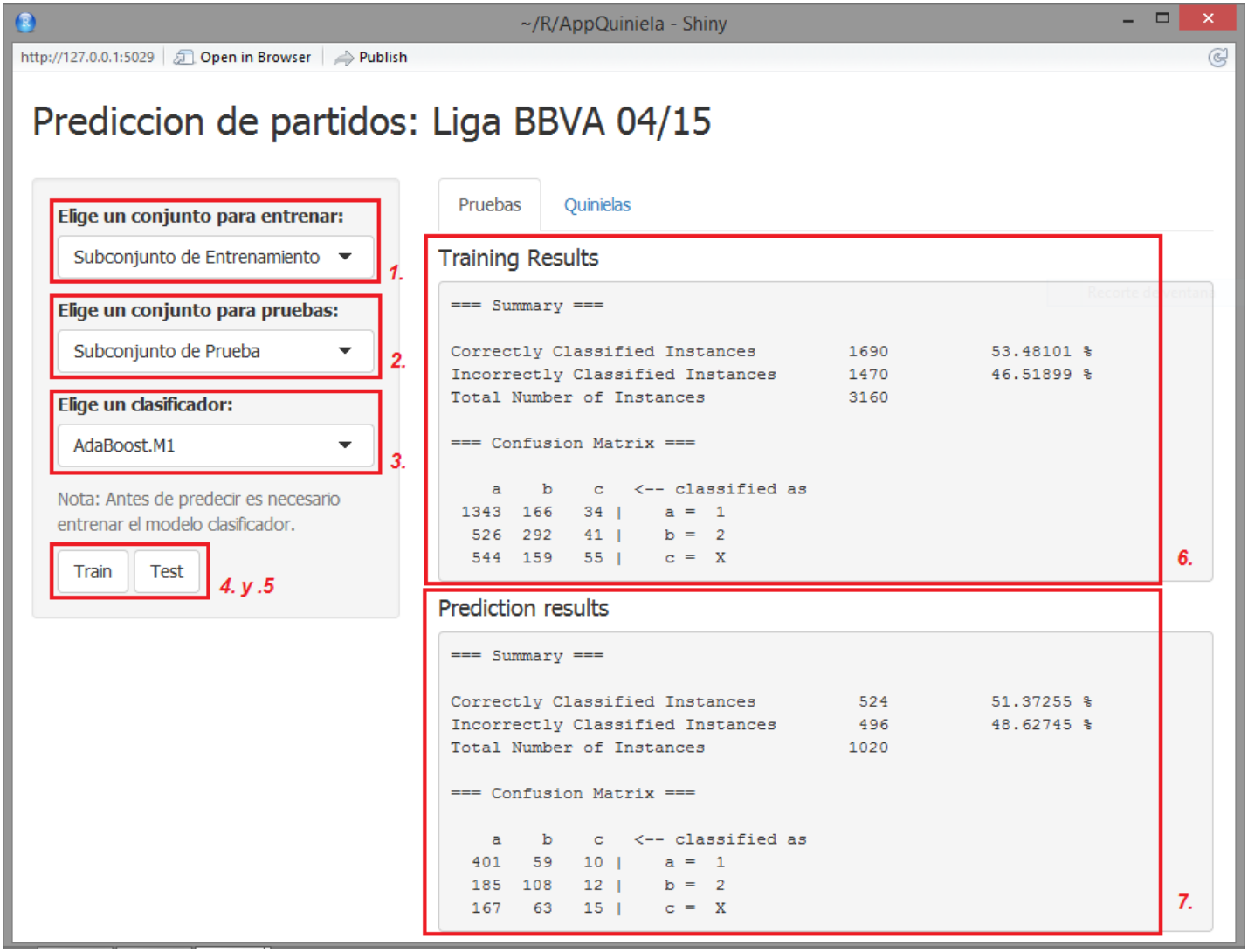

Figura 17. Ejemplo de ejecución de la aplicación AppQuiniela - Pruebas

Como ya se ha mencionado, para acceder a la segunda pestaña de la aplicación hemos de entrenar antes alguno de las técnicas y métodos de aprendizaje implementados en la pestaña *Pruebas*. En esta segunda pestaña, tenemos una tabla en la que podemos observar los partidos jugados en una cierta jornada de una temporada, parámetros que podemos cambiar para poder observar otros partidos. De ellos vemos la temporada y la jornada del encuentro, los equipos que competían en el partido, el resultado real y el resultado predicho por la técnica que previamente hubiéramos entrenado, por ello la importancia de entrenar antes de acceder a esta. Se le ha puesto el nombre de *Quinielas* porque los partidos que se juegan en una jornada de la primera división de española de fútbol representan dos tercios de los partidos que constituyen el juego de "*Loterías y Apuestas del Estado"*, "*La Quiniela"*.

En la aplicación se puede (Fig. 18):

1. Elegir la temporada en la que se jugaron los encuentros.
2. Elegir la jornada en la que se jugaron los encuentros.
3. Visualizar los datos básicos del encuentro seleccionado.
4. Visualizar los resultados reales de los encuentros seleccionados.
5. Visualizar los resultados predichos por el modelo para los encuentros.

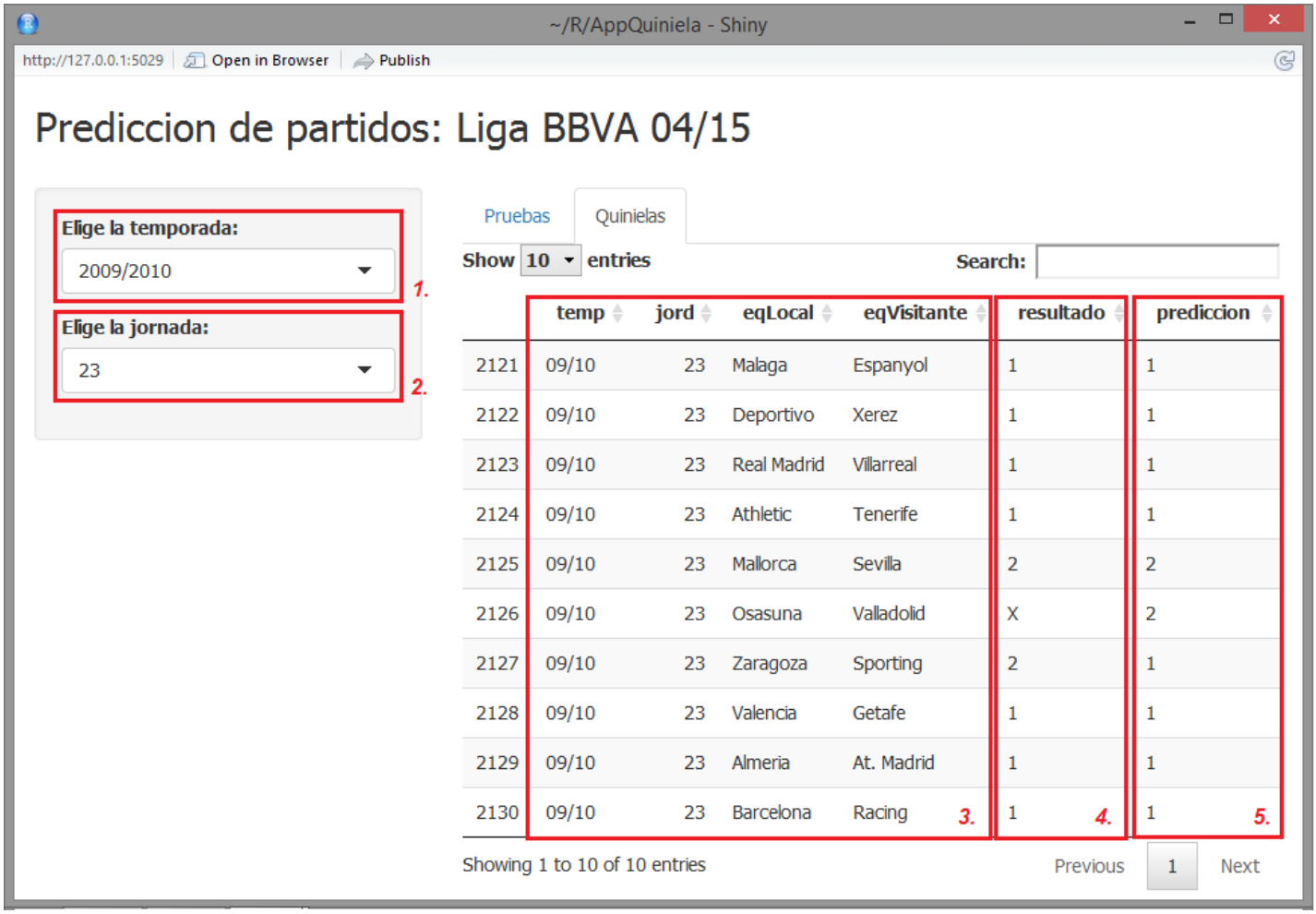

Figura 18. Ejemplo de ejecución de la aplicación AppQuiniela - Quinielas